Deckhouse Platform Administrator's guide

Version latest

Generated: November 21, 2026

Contents

Contents 2
Deckhouse Platform documentation 90
System requirements 90
- Deployment options 90
- Resource requirements 91
- Node requirements 94
- Network requirements 94
- Requirements for running virtual machines 95
- Access to the infrastructure 95
- Additional resources 95

Installation 95
- Installation methods 96
- Installation options 96
- Installation requirements 96
- Preparing the Configuration 96
  - Installation configuration file 96
  - Post-bootstrap script 103
- Installing 104
  - Pre-installation checks 105
  - The local deckhouse system user 109
  - Aborting the installation 109
- Installing virtualization 110
  - Placement of virtualization components across nodes 110
  - Cluster with taints on all nodes 111
- Air-gapped environment, working via proxy and using external registries 111
  - Installing from an external (third-party) registry 112
  - Nexus configuration notes 113
  - Harbor configuration notes 119
  - Manual loading of Deckhouse Platform images and vulnerability DB into a private registry 120
  - Creating a cluster and running Deckhouse Platform without using release channels 124
  - Using a proxy server 124
  - Automatic proxy variable loading for users in CLI 125
- Additional resources 126

How to configure? 126
- Deckhouse configuration 126
  - Modifying cluster configuration 127
    - Modifying protected parameters 128
      - Modifying protected parameters using dhctl 128
      - Manually changing protected parameters 129
  - Viewing current configuration 129
- Configuring the module 130
  - Enabling and disabling the module 130
- Module bundles 131
  - Things to keep in mind when working with the Minimal module set 132
  - Accessing documentation for the current version 132
- Managing placement of Deckhouse components 132
  - Advanced scheduling 132
  - Module features that depend on its type 133
- Additional resources 133

DP components registry 134
- Additional resources 134

Managing the registry in Deckhouse Platform-managed clusters 134
- Implementations of container image pull path management 134
- Current implementation 135
  - Modes of operation 135
  - Installer behavior when no container image registry is configured 136
  - Garbage collection 136
    - Running garbage collection manually 137
  - Additional container image registries 138
  - Requirements for using the current implementation 138

Examples of configuring the current implementation 138
Configuring pull path management for Deckhouse Platform component images 138
Enabling the in-cluster cache 139
Switching the cluster into an air-gapped state 140
Adding an additional container image registry 141
Pulling from a private container image registry without declaring it 141
Turning off pull path management 142
V1 implementation 142
Restrictions on Deckhouse Platform component registry settings 143
Restrictions during cluster installation 143
Operating conditions restrictions 143
Restrictions on mode switching 143
Examples of switching between the v1 implementation’s modes 143
Switching to Unmanaged mode 143
Migrating from the v1 implementation to the current one 145
Determining the migration method and steps 145
Checking which implementation the cluster uses 146
Migrating from Unmanaged mode 146
Migrating from Direct mode 146
Migrating from Proxy mode 147
Migrating an air-gapped cluster from Local mode 148
Migration to registry management format using the registry module 149
For containerd v2 149
For containerd v1 150
Migration to the deprecated registry management format (without the registry module) 154
Checking the registry mode switch status in the v1 implementation 156
Additional resources 158
Switching a Managed Kubernetes cluster to use third-party registry 158
Additional resources 160
Authentication 160
Additional resources 160
Integration with external authentication providers 161
General integration workflow 161
Limiting the identities of a provider 162
Checking provider connectivity 163
OIDC (OpenID Connect) integration 164
Keycloak 164
Blitz Identity Provider 165
Okta 166
SAML 2.0 integration 166
LDAP integration 167
GitHub integration 168
GitLab integration 169
Atlassian Crowd integration 169
Bitbucket Cloud Integration 170
Additional resources 170
Local authentication 171
Creating a static user 171
Deleting a user 172
Local user operations 172
Administrative operations 172
Self-service operations 173
Creating UserOperation manually 173
Viewing account state 174
Adding a user to a group 174
Configuring password policy 175
Configuring two-factor authentication (2FA) 176
Additional resources 176
Accessing the Kubernetes API via load balancer 176
How API access protection works in Kubernetes 177
Access using Basic Authentication (LDAP) 178
Kerberos (SPNEGO) SSO for LDAP 178

Additional resources 179
Resource configuration 180
Example configuration 180
Additional resources 181
Authorization 181
Combined use of ClusterAuthorizationRule, AuthorizationRule, and RBAC 182
Who gets access and when 182
User authentication 182
Service account authentication 182
Additional resources 185
Basic authorization model 185
High-level roles used in the basic model 186
Default access list for each high-level role 187
AuthorizationRule example 192
ClusterAuthorizationRule example 192
How rules are applied 193
Extending permissions for high-level roles 193
Additional resources 194
Granular authorization model 194
Role scopes 195
Access levels 195
Namespace roles 196
Automatic access to cluster-wide catalog resources 196
Project roles 197
System and subsystem roles 197
Access of system and subsystem roles to the namespaces of the modules 198
Who can assign roles 198
Global resource dictionaries 200
Automatic binding 200
Role model subsystems 200
How the roles are built: aggregation and capabilities 202
Creating a new subsystem role 203
Extending a custom role 204
Extending existing subsystem roles 205
Extending subsystem roles by adding a new namespace 206
Extending existing namespace roles 206
Creating a custom namespace or project role 207
Admin level restrictions and superadmin rights 208
Built-in protection mechanisms of the role model 209
Display names of roles 209
Deprecated role names 210
Custom roles of the previous scheme 211
Getting an equivalent of the basic model’s ClusterAdmin and SuperAdmin roles 211
Additional resources 212
Cluster admin access model 213
Kubeconfig files on master nodes 213
RBAC-based admin access 213
Recommended admin access 213
Root kubeconfig symlink 213
Security hardening 214
Break-glass access 214
Additional resources 214
Granting permissions to users and service accounts 214
Granting permissions using AuthorizationRule and ClusterAuthorizationRule (basic role model) 215
Granting permissions to a user within a single namespace 215
Granting permissions to a user in all namespaces 215
Granting permissions using ClusterRoleBinding and RoleBinding (granular role model) 216
Assigning cluster administrator permissions (granular role model) 216
Assigning networking administrator permissions (granular role model) 217
Granting administrator permissions to a user within a namespace (granular role model) 218
Additional resources 219

Checking user permissions 220
Quick check of a single permission 220
Checking whether a user has access 220
Getting a full permission report 221
Additional resources 222
Access for CI/CD 222
ServiceAccount 223
Prerequisites 223
Create ServiceAccount and a long-lived token 223
Grant permissions 223
Get API URL and CA certificate 224
Create kubeconfig 225
Short-lived tokens (TokenRequest) 225
Usage patterns in CI 225
Basic Auth 226
Prerequisites 226
Enable 226
Get API endpoint 226
Verification 226
Configure kubeconfig 226
Use in GitLab CI 227
Token Exchange 227
Prerequisites 227
Create DexClient 227
Create TokenExchangeProvider 228
Get Dex and API URLs 229
Grant RBAC 229
Token exchange request 230
GitLab CI 230
GitHub Actions 231
Migration from DexProvider 231
Diagnostics 232
Additional resources 233
Resource listing with access-based filtering 233
Using via Deckhouse CLI 233
Resource filtering scopes 233
Examples 233
Example of an interface query for where the user may perform actions 234
Using from your own code (HTTP API) 235
How it works 236
Aggregated APIs 237
Security model 238
Special case: namespaces 238
Special case: projects 238
Monitoring 238
Components involved 239
Additional resources 239
Integration with IaaS 240
Public cloud providers 240
Connection and authorization in Yandex Cloud 241
Preparing the environment 241
Creating a service account 241
Assigning IAM roles 242
Generating an authorization key 242
Credentials Secret 242
Checking and increasing quotas 243
Reserving a public IP 243
Additional resources 243
Layouts and configuration in Yandex Cloud 243
Layouts 243
Standard 243
WithoutNAT 245

WithNATInstance 247
Configuration 249
Configuration example 249
Network parameters and security 251
Internal node addressing 251
Assigning external IP addresses and outgoing traffic 251
DNS and DHCP settings for internal networks 252
Using pre-existing subnets 253
Additional external networks 253
Configuring security groups in Yandex Cloud 254
Configuring access via bastion host 255
Replacing the OS image 256
Additional resources 256
Storage and load balancing in Yandex Cloud 256
Storage (CSI and StorageClass) 256
Excluding unnecessary StorageClasses 257
Creating additional StorageClasses and block size 257
Setting the default StorageClass 258
Resizing a PVC 258
Load balancing 261
External LoadBalancer 261
Internal LoadBalancer 261
Service annotations 261
Using a separate target group for a NodeGroup 262
Target group health checks 263
Applying changes 263
Changing networkType does not recreate CloudEphemeral nodes 264
Integrating manually created VMs 264
Additional resources 264
Integration with Yandex Cloud services 264
Integration with Yandex Lockbox 265
Integration with Yandex Managed Service for Prometheus 267
Additional resources 268
Connection and authorization in VK Cloud 268
Layouts and configuration in VK Cloud 270
List of required OpenStack services 270
Layouts 270
Standard 270
StandardWithNoRouter 272
Simple 274
SimpleWithInternalNetwork 276
Configuration 278
Configuration examples 279
Example 1 279
Example 2 279
LoadBalancer configuration 280
IngressNginxController with a pre-created load balancer 280
IngressNginxController with a pre-allocated floating IP 280
Uploading an image to VK Cloud 281
Replacing the OS image 282
Verifying security group support by the cloud provider 282
Cinder CSI resilience during re-authentication in OpenStack 282
Configuring online volume resize 283
Using rootDiskSize 283
Disks in VK Cloud 283
rootDiskSize parameter 283
Recommendations for master nodes and bastion with network-attached disks 283
Recommendations for ephemeral nodes with local disks 284
Checking the disk size in a flavor 284
Overriding the cloud provider’s default disk type 284
Connection and authorization in Selectel Cloud 284
Layouts and configuration in Selectel Cloud 284

- List of required OpenStack services 284
- Layouts 285
  - Standard 285
  - StandardWithNoRouter 287
  - Simple 288
  - SimpleWithInternalNetwork 290
- Configuration 292
  - Configuration examples 293
    - Example 1 293
    - Example 2 293
  - How to create preemptible nodes in Selectel 294
  - LoadBalancer configuration 294
    - IngressNginxController with a pre-created load balancer 295
    - IngressNginxController with a pre-allocated floating IP 295
  - Uploading an image to Selectel 296
  - Replacing the OS image 297
  - Verifying security group support by the cloud provider 297
  - Cinder CSI resilience during re-authentication in OpenStack 297
  - Configuring online volume resize 298
  - Using rootDiskSize 298
    - Disks in Selectel 298
    - rootDiskSize parameter 298
      - Recommendations for master nodes and bastion with network-attached disks 298
      - Recommendations for ephemeral nodes with local disks 298
    - Checking the disk size in a flavor 299
  - Overriding the cloud provider’s default disk type 299
  - Offline disk resizing 299
- Connection and authorization in OVH Cloud 299
- Layouts and configuration in OVH Cloud 299
  - List of required OpenStack services 300
  - Layouts 300
    - Standard 300
    - StandardWithNoRouter 302
    - Simple 304
    - SimpleWithInternalNetwork 305
  - Configuration 307
    - Configuration examples 308
      - Example 1 308
      - Example 2 308
    - LoadBalancer configuration 309
      - IngressNginxController with a pre-created load balancer 309
      - IngressNginxController with a pre-allocated floating IP 309
    - Uploading an image to OVHcloud 310
    - Replacing the OS image 311
    - Verifying security group support by the cloud provider 311
    - Cinder CSI resilience during re-authentication in OpenStack 311
    - Configuring online volume resize 312
    - Using rootDiskSize 312
      - Disks in OVHcloud 312
      - rootDiskSize parameter 312
        - Recommendations for master nodes and bastion with network-attached disks 312
        - Recommendations for ephemeral nodes with local disks 313
      - Checking the disk size in a flavor 313
    - Overriding the cloud provider’s default disk type 313
    - Offline disk resizing 313
- Connection and authorization in OpenStack 313
- Layouts and configuration in OpenStack 314
  - List of required OpenStack services 314
  - Layouts 314
    - Standard 314
    - StandardWithNoRouter 316
    - Simple 318

SimpleWithInternalNetwork 320
Configuration 322
Configuration examples 323
Example 1 323
Example 2 323
LoadBalancer configuration 324
IngressNginxController with a pre-created load balancer 324
IngressNginxController with a pre-allocated floating IP 324
Uploading an image to OpenStack 325
Replacing the OS image 326
Verifying security group support by the cloud provider 326
Cinder CSI resilience during re-authentication in OpenStack 326
Configuring online volume resize 327
Using rootDiskSize 327
Disks in OpenStack 327
rootDiskSize parameter 327
Recommendations for master nodes and bastion with network-attached disks 327
Recommendations for ephemeral nodes with local disks 328
Checking the disk size in a flavor 328
Overriding the cloud provider’s default disk type 328
Offline disk resizing 328
Connection and authorization in Amazon Web Services 328
Requirements 328
Accessing the AWS API 329
Configuring IAM via the AWS web console 331
Configuring IAM via AWS CLI 331
Configuring IAM via Terraform 332
Using a custom IAM role for nodes (iamNodeRole) 333
Additional resources 333
Layouts and configuration in Amazon Web Services 333
WithoutNAT 333
WithNAT 335
Defining AWSClusterConfiguration 336
Internal addressing and subnets 336
Security groups 337
Setting up VPC peering 337
Configuring access via bastion host 337
Creating a cluster in a new VPC with an existing bastion host 338
Creating a cluster in a new VPC and deploying a bastion host 338
Using an existing VPC (existingVPCID) 338
Preemptible nodes 338
Replacing the OS image 338
Additional resources 339
Storage and load balancing in Amazon Web Services 339
Storage (CSI and StorageClass) 339
Expanding volume size 339
Load balancing 339
Configuring LoadBalancer when Ingress nodes are not in all zones 340
Connecting CloudStatic nodes 340
Additional resources 340
Integration with Amazon Web Services 340
Working with Amazon Elastic Container Registry (ECR) 340
Additional resources 341
Connection and authorization in Microsoft Azure 341
Requirements 341
Accessing the Azure API 342
Additional resources 342
Layouts and configuration in Microsoft Azure 342
Standard 342
Replacing the OS image 343
Additional resources 343
Storage and load balancing in Microsoft Azure 343

- Storage 343
- Load balancing 344
- Additional resources 344

Integration with Microsoft Azure services 344
- Support for Service Endpoints 344
- Additional resources 345

Connection and authorization in Google Cloud Platform 345
- Creating a service account 345
  - Setup via Google Cloud console 345
  - Setup via Google Cloud CLI 345
- Using the service account 346
- Additional resources 346

Layouts and configuration in Google Cloud Platform 347
- Layouts 347
  - Standard 347
  - WithoutNAT 348
- Configuration 350
  - Enabling nested virtualization 351
  - Adding additional disks 351
  - Adding CloudStatic nodes to the cluster 352
- Preemptible nodes 352
- Replacing the OS image 352
- Additional resources 352

Private cloud providers 352

Connection and authorization in VK Cloud 352

Layouts and configuration in VK Cloud 353
- List of required OpenStack services 353
- Layouts 353
  - Standard 353
  - StandardWithNoRouter 355
  - Simple 357
  - SimpleWithInternalNetwork 359
- Configuration 361
  - Configuration examples 362
    - Example 1 362
    - Example 2 362
  - LoadBalancer configuration 363
    - IngressNginxController with a pre-created load balancer 363
    - IngressNginxController with a pre-allocated floating IP 363
  - Uploading an image to VK Private Cloud 364
  - Replacing the OS image 365
  - Verifying security group support by the cloud provider 365
  - Cinder CSI resilience during re-authentication in OpenStack 365
  - Configuring online volume resize 366
  - Using rootDiskSize 366
    - Disks in VK Private Cloud 366
    - rootDiskSize parameter 366
      - Recommendations for master nodes and bastion with network-attached disks 366
      - Recommendations for ephemeral nodes with local disks 367
    - Checking the disk size in a flavor 367
  - Overriding the cloud provider’s default disk type 367

Connection and authorization in OpenStack 367

Layouts and configuration in OpenStack 367
- List of required OpenStack services 367
- Layouts 368
  - Standard 368
  - StandardWithNoRouter 370
  - Simple 371
  - SimpleWithInternalNetwork 373
- Configuration 375
  - Configuration examples 376
    - Example 1 376

Example 2 376
LoadBalancer configuration 377
IngressNginxController with a pre-created load balancer 377
IngressNginxController with a pre-allocated floating IP 377
Uploading an image to OpenStack 378
Replacing the OS image 379
Verifying security group support by the cloud provider 379
Cinder CSI resilience during re-authentication in OpenStack 379
Configuring online volume resize 380
Using rootDiskSize 380
Disks in OpenStack 380
rootDiskSize parameter 380
Recommendations for master nodes and bastion with network-attached disks 380
Recommendations for ephemeral nodes with local disks 381
Checking the disk size in a flavor 381
Overriding the cloud provider’s default disk type 381
Offline disk resizing 381
Connection and authorization in Huawei Cloud 381
Requirements 381
Accessing the Huawei Cloud API 381
Creating a user group 382
Creating an IAM user 382
JSON policies 382
Layouts and configuration in Huawei Cloud 384
List of required Huawei Cloud services 384
Standard 385
VpcPeering 386
Additional resources 387
Storage and load balancing in Huawei Cloud 387
Storage 387
Load balancing 387
Additional resources 387
Integration with Huawei Cloud services 387
Working with InstanceClass 388
Replacing the OS image 388
Additional resources 388
Connection and authorization in Basis Dynamix 388
Requirements 388
Preparing a cloud image 388
Configuring access to the portal 389
Additional resources 389
Layouts and configuration in Basis Dynamix 389
Standard 389
StandardWithInternalNetwork 390
Required parameters 391
Additional resources 391
Storage and load balancing in Basis Dynamix 391
Storage 391
Changing the storage policy 392
Load balancing 392
Configuring a Service of the LoadBalancer type 392
Additional resources 392
Integration with Basis Dynamix cloud services 392
Key capabilities 392
Recommendations 393
Additional resources 393
Virtualization systems 393
Integration with built-in virtualization 393
Load balancers 393
Load balancer network policy 394
Additional resources 394
Connection and authorization in built-in virtualization 394

- Creating a user 395
- Assigning a role 395
- Generating a kubeconfig 395
- Credentials Secret 396

Layouts and configuration in built-in virtualization 396
- Standard 396
- Configuration 398
- Replacing the OS image 400
- Additional resources 401

Connection and authorization in VMware Cloud Director 401
- Resource preparation 401
  - User permissions 401
  - Adding a network 404
  - DHCP setup 407
  - Adding a vApp 409
  - Adding a network to a vApp 410
  - Incoming traffic 412
  - Configuring DNAT/SNAT rules on the Edge Gateway 412
  - Let’s Encrypt certificate issuance 413
  - Firewall setup 413
- Virtual machine template 415
  - Preparing the template from an OVA file 416
  - Configuring the template in VCD 420
  - Replacing the OS image 425
- Storage usage 426
- Using the LoadBalancer 426
- Additional resources 426

Layouts and configuration in VMware Cloud Director 426
- List of required VMware Cloud Director services 426
- Layouts 426
  - Standard 426
  - WithNAT 427
- Configuration 430
  - Example configuration of VCDInstanceClass 431
  - Storage 431
    - CSI 431
- Additional resources 431

Connection and authorization in VMware vSphere 432
- Requirements 432
  - VM image requirements 432
  - Preparing the virtual machine image 432
  - Replacing the OS image 433
- vSphere configuration 433
  - Configuration in vSphere Client 433
    - Creating tags and tag categories in vSphere Client 433
    - Configuring Datastore in vSphere Client 436
    - Creating and assigning a role in vSphere Client 437
  - Configuration with govc 438
    - Installing govc 438
    - Creating tags and tag categories with govc 438
    - Datastore configuration with govc 439
    - Creating and assigning a role with govc 439
- Additional resources 440

Layouts and configuration in VMware vSphere 440
- Standard 440
- List of required privileges 442
- Additional resources 445

Storage and load balancing in VMware vSphere 445
- Storage 445
  - Resizing a volume (PVCs) 445
- Load balancing 445
- CSI 446

- Important information on PVC size expansion 446
- Datastore configuration 446
- Additional resources 446
- Integration with VMware vSphere services 446
- vSphere resource management 447
  - Removing CloudPermanent nodes in vSphere 447
    - Reducing the number of nodes 447
    - Completely removing a node group 447
- Additional resources 448
- Connection and authorization in zVirt 448
- Requirements 448
- User permissions 449
- Preparing a cloud image 449
- Preparing a virtual machine template 450
- Replacing the OS image 452
- Credentials Secret 452
- Additional resources 453
- Layouts and configuration in zVirt 453
- Layouts 453
  - Standard 453
- Configuration 454
  - Configuration example 455
  - Static network configuration of nodes 456
- Additional resources 457
- Storage and load balancing in zVirt 457
- Storage 457
  - StorageClasses for storage domains 457
  - Virtual machine disks 457
- Load balancing 457
- Additional resources 458
- Integration with zVirt services 458
- Key features 458
- Additional resources 458
- Hybrid clusters 458
- NodeGroup types 458
- Ways to add nodes 459
- General network requirements 459
- Additional resources 459
- Hybrid cluster with Yandex Cloud 459
- Prerequisites 459
- Adding automatically created nodes 460
- Adding manually created nodes through a bootstrap script 463
- Additional resources 465
- Hybrid cluster with VCD 465
- Prerequisites for VCD 465
- Adding automatically created nodes 465
- Adding manually created nodes through a bootstrap script 466
- Additional resources 468
- Hybrid cluster with vSphere 468
- Prerequisites for vSphere 468
- Adding automatically created nodes 468
- Adding manually created nodes through a bootstrap script 470
- Additional resources 472
- Hybrid cluster with built-in virtualization 473
- Prerequisites 473
- Adding automatically created nodes 473
- Adding manually created nodes using a bootstrap script 477
- Additional resources 478
- High reliability and availability 479
- Additional resources 479
- Recommendations for configuring cluster nodes and preventing overload 479
- Master nodes 479

- Frontend nodes 479
- Monitoring nodes 480
- System nodes 480
- Preventing Kubernetes API overload (FlowSchema) 480
- Delaying node reboot or shutdown while critical pods are running 480
  - Enabling a node shutdown delay 480
  - Searching pods that prevent shutdown and performing the node troubleshooting 481
- Additional resources 481

Managing HA mode 481
- Enabling HA mode globally 482
  - Using ModuleConfig/global custom resource 482
  - Using Deckhouse web UI 482
- Configuring HA mode with two master nodes and an arbiter node 482
  - Configuring in a cloud cluster 482
  - Configuring in a static cluster 484
- Enabling HA mode for individual components 484
- Additional resources 485

Chaos engineering 485
- Additional resources 486

Virtual machine fault tolerance and balancing 486
- VM rebalancing 486
- Diagnosing a slow VM 487
- ColdStandby 487
- Additional resources 488

Platform scaling and node management 488

General management and configuration of the control plane 489
- Main features 489
- Enabling, disabling, and configuring the module 489
  - Enabling / disabling 489
  - Configuration 490
  - Checking DP status and queues 490
- Certificate management 491
  - PKI management 491
  - Additional SANs and auto-update 491
  - Kubelet certificate rotation 491
  - Manual renewal of control plane certificates 492
- Speeding up pod restarts after losing connection to a node 492
- Forcibly disabling IPv6 on cluster nodes 493
- etcd periodic defragmentation configuration 493
- Additional resources 494

Control plane recovery and debugging 494
- Recovery from failures 494
- Restoring etcd cluster functionality 494
- Restoring a master node when kubelet fails to load control plane components 494
  - containerd 495
- etcd restore 495
  - Viewing etcd cluster members 495
    - Option 1 495
    - Option 2 495
  - Restoring the etcd cluster in case of complete unavailability 495
  - Recovering etcd after panic: unexpected removal of unknown remote peer error 496
  - Actions to take when etcd database exceeds quota-backend-bytes limit 496
- etcd defragmentation 497
  - How to defragment an etcd node in a single-master cluster 498
  - How to defragment etcd in a cluster with multiple master nodes 499
- High availability 501
- Additional resources 501

Scaling and changing master nodes 501
- Scaling and single/multi-master transition 501
  - Control plane operation modes 502
  - Automatic scaling of master nodes 502
  - Common scaling scenarios 502

Removing the master role from a node without deleting the node itself 503
Changing the OS image of master nodes in a multi-master cluster 503
In a cloud cluster 504
In a static cluster 505
Changing the OS image in a single-master cluster 507
In a cloud cluster 507
In a static cluster 507
Adding master nodes to a static or hybrid cluster 507
Adding master nodes in a cloud cluster 508
Reducing the number of master nodes in a cloud cluster 510
Accessing the DP controller in a multi-master cluster 511
Additional resources 511
Node management basics in Deckhouse 511
Automatic deployment and updates 512
Basic node and OS configuration 512
Disruptive updates 513
Example of a system NodeGroup 513
Example NodeGroupConfiguration descriptions 513
Installing the cert-manager plugin for kubectl on master nodes 513
Werf configuration for ignoring the Ready status of a node group 514
Settings for Static and CloudStatic NodeGroups 514
Running DP on an arbitrary node 515
Additional resources 515
Adding and managing cloud nodes 515
Adding CloudEphemeral nodes in a cloud cluster 515
Configuration for CloudEphemeral NodeGroups 516
Modifying the cloud provider configuration in a cluster 516
NodeGroup autoscaling 517
Example autoscaling scenario 517
Allocating nodes for specific workloads 518
Accelerating node provisioning in the cloud during horizontal application scaling 518
Example NodeGroup definition 519
Cloud nodes 519
Configuring CloudEphemeral nodes using NodeGroupConfiguration 519
Example NodeGroupConfiguration definitions 519
Setting a sysctl parameter 519
Adding a root certificate to the host 520
Adding a certificate to the OS and containerd 521
Kernel update on nodes 523
Debian-based distros 523
CentOS-based distros 523
Adding CloudPermanent nodes to a cloud cluster 524
Adding a CloudStatic node to a cluster 526
Manually 526
Using the Cluster API Provider Static 526
Using Cluster API Provider Static for multiple node groups 528
Adding master nodes in a cloud cluster 529
Using NodeGroup with priority 530
NodeGroup states and their interpretation 530
Additional resources 532
Adding and managing bare-metal nodes 532
Adding nodes to a bare-metal cluster 532
Manual method 532
Automatic method 533
Creating nodes through BMC 534
Modifying a static cluster configuration 535
Moving a static node between NodeGroups 535
Manual cleanup of a static node 535
NodeGroup example 535
Example NodeGroup definition for static nodes 536
Changing the CRI for a NodeGroup 536
Changing the NodeGroup of a static node 536

- Changing the IP address in a StaticInstance 537
- Deleting a StaticInstance 537
- Additional resources 537

Custom node configuration 537
- Monitoring script execution 539
- Script re-execution mechanism 540
- Script writing specifics 540
- How to use containerd with Nvidia GPU support 540
- How to deploy a custom containerd configuration file 541
- Adding a configuration for an additional registry 542
  - Old Method 543
    - Adding authorization to an additional registry (deprecated method) 543
    - Configuring a certificate for an additional registry (deprecated method) 544
    - Adding TLS skip verify (deprecated method) 545
    - Configuring a mirror for access to public registries (deprecated method) 546
  - New Method 547
    - Adding authorization to an additional registry (actual method) 548
    - Configuring a certificate for an additional registry (actual method) 548
    - Adding TLS skip verify (actual method) 549
    - Configuring a mirror for access to public registries (actual method) 550
- How to automatically put custom labels on the node 551
- Additional resources 551

Migrating container runtime to containerd v2 552
- Requirements 552
- How to enable containerd v2 553
- Additional resources 553

Maintenance of nodes running virtual machines 553
- VM migration and node maintenance 554
  - Moving a selected VM to another node 554
  - Dedicated migration network 555
  - Checking VMs before node maintenance 555
  - Maintenance mode 556
  - Restarting virtual machines during node maintenance 556
  - Shutting down and rebooting a node with virtual machines 557
- Additional resources 558

Application scaling and resource management 558

Scaling by metrics 559
- Scaling based on metrics 559
  - Configuration 559
  - Configuring metrics using observability module resources 560
  - Working with unstable metrics 561
  - Retrieving metric values 561
  - Configuring autoscaling in the Deckhouse Web Interface 561
- Additional resources 562

Horizontal pod autoscaling 562
- How Horizontal Scaling (HPA) works 562
- Available metric types for HPA 562
- HPA recommendations 563
- HPA limitations 563
- How to enable or disable HPA 563
- HPA configuration 563
- HPA configuration examples 564
  - Scaling based on CPU and memory 564
  - Scaling based on object metrics 565
  - Scaling based on external metrics 565
- Additional resources 566

Vertical pod autoscaling 566
- How Vertical Scaling (VPA) works 566
- How to enable or disable VPA 566
- VPA configuration 567
  - Module configuration example 567
- Additional resources 568

Pod restart on configuration change 568
Supported annotations 568
Usage examples 569
Tracking all changes in all attached resources 569
Tracking changes in specific resources 570
Tracking changes in resources from the list 570
Enabling or disabling pod restart 571
Configuration 571
Additional resources 572
Scheduler 572
Pod scheduling 572
General overview of the scheduler 572
Pod scheduling phases 573
How to modify or extend scheduler logic 573
Additional pod placement mechanisms 574
Scheduler profiles 575
Pod redistribution 575
Considering pod Priority Classes 575
How DP considers pod priority 576
Which pods are not evicted 576
How to enable or disable pod eviction 576
Descheduling strategies 577
Additional resources 579
Priority classes 580
Available priority classes 580
Creating additional priority classes 581
Preemption mechanism 582
How it works 582
Protecting a pod from eviction 582
Differences between priority classes and other resource management mechanisms 582
Operations and diagnostics 583
Additional resources 583
Network 583
Configuring and managing incoming traffic 583
Managing outgoing traffic 584
Internal network configuration 584
Configuring inter-cluster communication 584
Network policies 584
Virtual machine networking 584
Additional resources 584
Internal network configuration 584
Internal network operation modes 585
Mode configuration 585
Services 585
Supported service types 586
Configuring the service network with kube-proxy 586
Additional resources 586
Encrypting traffic between pods 586
mTLS configuration example 587
Additional resources 590
Configuring cluster network stack visualization 591
Additional resources 591
Switching CNI from Flannel or Simple bridge to Cilium 591
Incoming traffic balancing 593
Network-level load balancing (NLB) 593
Application-level load balancing (ALB) 593
Difference between the Kubernetes Gateway API and an API gateway 593
Role separation in the Gateway API model 594
Choosing an ALB implementation 594
Comparison of the ingress-nginx and alb modules 594
Next steps 595
Additional resources 595

Balancing with MetalLB 596
Layer 2 mode 596
How it works 596
Advantages over classic MetalLB 596
Behavior comparison 596
Example of using MetalLB in L2 LoadBalancer mode 597
BGP mode 599
How it works 599
Configuration 599
Establishing BGP sessions 599
Assigning IP addresses to Services 599
Announcing IP addresses 599
Traffic distribution 599
Benefits of using BGP 599
Drawbacks of using BGP 600
Example of using MetalLB in BGP LoadBalancer mode 600
Running speakers on nodes with custom taints 600
Additional resources 601
Balancing in clusters on cloud platforms 601
Example of creating an Ingress controller for the OpenStack provider 602
Example of creating a ClusterIP-type Service 602
Example for VK Cloud 602
ALB with Ingress NGINX Controller 603
Traffic termination options 603
HTTPS termination 603
Monitoring and statistics 603
Statistics 604
Basic principles of collecting statistics 604
Metrics structure and representation 604
Load balancing configuration examples 604
Example for AWS (Network Load Balancer) 604
Example for GCP, Yandex Cloud, and Azure 605
Example for OpenStack 605
Example for VK Cloud 605
Example for bare metal 606
Example for bare metal with external load balancer 606
Example for bare metal (MetalLB in BGP LoadBalancer mode) 606
Example for bare metal (MetalLB in L2 LoadBalancer mode) 607
Example of segregating access between public and administrative zones 609
Specifics of using a single Ingress controller 609
Using separate Ingress controllers 610
Module support and security 612
Additional resources 612
ALB with Kubernetes Gateway API 612
Overview and scheme 613
Object roles 613
Why ListenerSet 613
ClusterALBInstance and ALBInstance 613
Validating configuration 614
Inlets 614
How to configure ALB 614
Steps before enabling 614
Enabling the module and creating a Gateway 615
Creating a ListenerSet 615
Creating routes 616
Publishing service domains 616
Selecting the default DP gateway when using multiple ClusterALBInstances 617
Changing the default DP gateway 617
Examples for different environments 617
Changing the inlet while keeping the current Gateway 619
Opening an additional TCP/UDP port on the cluster-wide gateway 620
Separating public and administrative zones 620

Namespaced publishing (ALBInstance) 621
Issuing TLS certificates with cert-manager 621
Using with other modules and third-party solutions 622
Alongside ingress-nginx 622
Alongside third-party Gateway API implementations 622
Enabling HTTP/3 623
Using GeoIP and GeoLite2 624
Using GeoIP Headers 625
Configuring OpenTelemetry tracing 625
Configuring OpenTelemetry tracing TLS 625
Diagnostics and verification 626
Additional resources 627
Migrating from ingress-nginx to alb 627
Reasons to move to the Gateway API 627
Model comparison 628
Ingress API and ingress-nginx 628
Gateway API and alb 628
Key architectural differences 629
Step 1. Preparing alb module infrastructure 629
Choosing ALBInstance or ClusterALBInstance 630
Inlet configuration 630
TLS and certificates 631
Step 2. Migrating DP interfaces 631
Step 3. Migrating application publishing 632
Converting Ingress to Gateway API 632
Using the built-in ingress2gateway tool 632
Extending Gateway API with annotations 633
Step 4. Switching traffic to the alb module 634
Testing the alb module via the Ingress NGINX Controller 634
Preserving HTTP-01 validation 636
Choosing the switching method 636
Automatically provisioned load balancer 636
Manually managed load balancer 637
Direct HostPort or HostWithFailover access 637
Validating the cutover 637
Rolling back 638
Step 5. Cleanup 638
Additional resources 638
ALB with Istio 638
Ingress to publish applications 639
Istio Ingress Gateway 639
Ingress NGINX 639
Additional resources 640
Ensuring high availability and fault tolerance (keepalived) 640
Module usage examples 640
Multiple public IP addresses 640
Switching keepalived manually 641
Additional resources 641
Egress Gateway configuration 641
Egress Gateway usage modes 642
Basic mode 642
Virtual IP mode 642
Configuration principle 642
Node maintenance 642
Comparison with CiliumEgressGatewayPolicy 642
Configuration examples 643
EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode) 643
EgressGateway in VirtualIPAddress mode 643
EgressGatewayPolicy 643
Additional resources 644
Inter-cluster cooperation (cluster alliance) 644
Service mesh usage specifics in DP 644

- Differences between federation and multicluster 644
- Models for combining multiple clusters (Istio example) 645
- Problems solved with multicluster and federation 645
- Additional resources 645

Multicluster 645
- Multicluster of Istio funds 645
  - Requirements for clusters 645
  - General principles 646
  - Enabling the multicluster 646
  - Managing the multicluster 646
  - Example of configuring a multicluster of two clusters 646
- Additional resources 646

Federation 647
- Federation with Istio (Service Mesh) 647
  - Requirements for clusters 647
  - General principles of federation 647
  - Enabling the federation — created services 647
  - Managing the federation 647
  - Example of configuring a federation of two clusters 648
- Additional resources 649

Network policies 649
- Network policy implementation in DP 649
- What is available in each engine 649
- Network infrastructure requirements 650
- Sections 650
- Additional resources 650

Kubernetes NetworkPolicy 650
- Isolation model 650
- Selectors and fields 651
  - AND vs OR in selectors 652
  - Selecting a namespace by name 652
  - Port ranges 652
- Default policies for a namespace 652
- Edge cases 653
  - hostNetwork pods 653
  - Pod lifecycle 653
  - Existing connections 653
  - L4 only 653
- Enforcement without Cilium: the network-policy-engine module 653
- API limitations 654
- Additional resources 654

CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy 654
- How rules are evaluated 654
- What CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy add 654
- Resource structure 655
  - Egress to a Kubernetes service 655
- Entities 656
- L4 rules: ICMP and SNI 656
  - ICMP and ICMPv6 656
  - TLS Server Name Indication (SNI) 657
- L7 rules 657
  - Kafka 658
  - DNS Policy and IP Discovery 658
- FQDN rules 659
- Deny rules 659
- Default policies via CiliumNetworkPolicy 660
- Audit mode (policyAuditMode) 660
- Additional resources 661

Host firewall on nodes 661
- How a host firewall differs from regular policies 661
- Safe rollout 661
- Mandatory rules 662

Example: API server access for the control plane 662
Example: SSH from administrative networks 662
Example: baseline for worker nodes 663
Additional resources 663
Common policy examples 663
Deny all external ingress to a namespace, allow internal traffic 664
Allow ingress from a specific namespace 664
Allow egress to DNS and one CIDR only 664
Allow specific pods to reach the API server 665
Allow only GET requests to an API at L7 665
Allow egress to specific DNS names (FQDN) 666
Deny access to the cloud metadata service 667
Additional resources 667
Diagnostics and observability 668
Inspecting an applied policy 668
Observability via Hubble 668
Continuous flow logs collection 669
Diagnosing FQDN rules 669
Common mistakes 669
DNS broken after default-deny egress 669
AND vs OR mixed up 669
Policy does not apply to hostNetwork pods 670
FQDN rule does not allow traffic 670
Connection drops after a policy change 670
“Policy not applied” checklist 670
Additional resources 670
Software-defined networking (SDN) 670
Additional resources 671
Configuring SDN in a cluster 671
Preparing the infrastructure for enabling the sdn module 671
Enabling the sdn module 671
Actions after enabling the sdn module 671
Marking interfaces for organizing additional software-defined networks 672
Combining multiple physical interfaces into a channel aggregation interface (bond interface) 672
Configuring and connecting additional virtual networks for use in application pods 674
Creating a publicly accessible network (cluster) 674
Creating a network based on tagged traffic 674
Creating a network based on direct interface access 675
Creating a project network (user network) 676
Checking the connection of an additional network to interfaces on nodes 676
IPAM: IP address pools for additional networks 677
Principles and features of IPAM in DP 677
Protection against conflicts when reusing IP addresses 678
Resources and settings for configuring IPAM 678
Example of assigning IP addresses to the cluster network 678
Configuring and connecting underlay networks for hardware device passthrough 679
Key features 679
Operation modes 679
Automatic interface grouping 679
Procedure for configuring and connecting physical interfaces to application subnets 679
Prerequisites for DPDK applications 680
Configuring hugepages 680
Configuring Topology Manager 680
Preliminary steps before creating an UnderlayNetwork 681
Checking and configuring network interfaces (NodeNetworkInterface) 681
Creating UnderlayNetwork in Dedicated mode 681
Creating UnderlayNetwork in Shared mode 682
Preparing namespaces for UnderlayNetwork usage 683
Configuring and connecting system networks (service networks) 684
Prerequisites for creating and using system networks 684
Creating a system network 684

VLAN type 684
Access type 685
SRIOVVirtualFunction type 685
Creating a pool of IP addresses for configuring the IPAM system network 685
Checking system network status 686
Additional resources 686
Virtual machine networking 686
Additional resources 687
Managing static routes and ip rules on cluster nodes 687
Configuration examples 687
Creating a route in main routing table 687
Creating a route in additional routing table 688
Creating an ip rule 688
Additional resources 689
Managing DNS in a Kubernetes cluster 689
Configuration example 690
Additional resources 690
Caching DNS requests on cluster nodes 690
Example custom DNS configuration in a Pod 690
Additional resources 691
BpfLB operation modes for external traffic processing 691
Additional resources 693
Synchronizing time on nodes 693
Enabling built-in time synchronization 693
Using custom NTP daemons 693
Additional resources 694
Access via OpenVPN 694
Configuration 694
External access to OpenVPN 694
Example for bare-metal clusters 695
Example for AWS and Google Cloud 695
Example for a public IP address on an external Load Balancer 695
Authentication 695
Available cluster resources after connecting to VPN 696
Auditing user connections 696
Why automatic DNS configuration doesn’t work on macOS and Linux with the OpenVPN client 696
How to revoke, rotate or delete a user certificate 696
How to rotate a server certificate 697
How to rotate a root certificate (CA) 697
Additional resources 697
Storage 698
Supported storage types 698
Software-defined storage 698
External storage 698
Key features 698
Storage configuration 698
Creating a StorageClass 699
Setting a default StorageClass 699
Additional resources 699
Configuring volume snapshot creation 699
Supported CSI drivers 699
Creating snapshots 699
Restoring from a snapshot 700
Additional resources 700
Local path provisioner storage 700
How it works 700
Limitations 701
Example LocalPathProvisioner resources 701
ReclaimPolicy: Retain (default) 701
ReclaimPolicy: Delete 701
Configuring Prometheus to use local storage 701
Additional resources 702

Node configuration for software-defined storage 702
Additional resources 702
Usage 702
Working with BlockDevice resources 702
Creating a BlockDevice resource 703
Criteria for device selection by the controller 703
Updating a BlockDevice resource 703
Deleting a BlockDevice resource 703
Working with LVMVolumeGroup resources 703
Creating an LVMVolumeGroup resource 703
Updating an LVMVolumeGroup resource 705
Deleting an LVMVolumeGroup resource 705
Removing a BlockDevice from an LVMVolumeGroup 706
Protecting against data leaks between volumes 706
Thick volume 706
Using Discard mode 706
Thin volume 707
Additional resources 707
Configuration scenarios 707
Cloning virtual machines 707
Methods and scenarios for configuring the disk subsystem of nodes 708
Storage with identical disks 709
Full mirror 709
Example of configuring SDS modules (identical disks, “Full mirror”) 710
Configuring the sds-local-volume module (identical disks, “Full mirror”) 710
Configuring the sds-replicated-volume module (identical disks, “Full mirror”) 710
Partial mirror 711
Example of configuring SDS modules (identical disks, “Partial mirror”) 712
Configuring the sds-local-volume module (identical disks, “Partial mirror”) 712
Configuring the sds-replicated-volume module (identical disks, “Partial mirror”) 713
Combined storage 714
Configuring additional disks (Full mirror) 715
Example of configuring SDS modules (combined storage, “Full mirror”) 715
Configuring the sds-local-volume module (combined storage, “Full mirror”) 715
Configuring the sds-replicated-volume module (combined storage, “Full mirror”) 716
Configuring additional disks (Partial mirror) 717
Example of configuring SDS modules (Combined storage, “Partial mirror”) 718
Configuring the sds-local-volume module (combined storage, “Partial mirror”) 718
Configuring the sds-replicated-volume module (combined storage, “Partial mirror”) 719
Additional resources 721
FAQ 722
Reasons why BlockDevice resources aren’t created in the cluster 722
Reasons why LVMVolumeGroup resources aren’t created in the cluster 722
Reasons why after deleting the LVMVolumeGroup, the resource and Volume Group remain 722
Reasons for failed attempts to create a Volume Group using the LVMVolumeGroup resource 722
Behavior of the LVMVolumeGroup resource when one of the devices in the Volume Group is disabled 722
Transferring management of an existing Volume Group on the node to the controller 723
Disabling tracking of the LVM Volume Group by the controller 723
Reasons for automatic setting of the tag storage.deckhouse.io/enabled=true on the Volume Group 723
Using the LVMVolumeGroupSet resource to create LVMVolumeGroup 723
Labels added by the controller to BlockDevice resources 723
Additional resources 724
Setting up local storage based on LVM 724
Configuring local storage 724
Quick start 724
Enabling modules 724
Preparing nodes for storage creation 725
Configuring storage on nodes 725
Selecting a volume cleanup method after PV deletion 728
Thick volumes 729

Thin volumes 729
System requirements and recommendations 729
Additional resources 729
Managing local storage based on LVM 729
Selecting specific nodes for module usage 729
Verifying PVC creation on the selected node 730
Removing a node from the module management 730
Verifying dependent LVMVolumeGroup resources on the node 731
Remaining sds-local-volume-csi-node Pod after removing labels 732
Absence of component service Pods on the desired node 732
Data migration between PVCs 733
Creating volume snapshots 734
Additional resources 735
Setting up replicated storage based on DRBD 735
Enabling the module 735
Discovery of LVM components 735
Connecting DRBD 736
Node pre-configuration 736
Creating LVM volume groups 736
Creating replicated thick pools 738
Creating replicated thin pools 738
Creating StorageClass objects 739
Additional resources 740
Managing DRBD-based replicated storage 741
System requirements 741
Recommendations 741
Retrieving space-usage information 741
Assigning the default StorageClass 741
Adding an existing LVMVolumeGroup 741
Changing DRBD volume limits and cluster ports 742
Correctly rebooting a node with DRBD resources 742
Moving resources to free up space in a Storage Pool 742
Automatic management of replicas and monitoring of LINSTOR state 743
Evicting DRBD resources from a node 743
Preparing and running the script 743
Example of removing a node from LINSTOR and Kubernetes 744
Example of removing resources from a node 744
Parameters of the evict.sh script 744
Troubleshooting 745
Start error of linstor-node while loading the DRBD module 745
FailedMount error when starting a Pod 745
Pod stuck in ContainerCreating 745
Input/output error 745
After the ReplicatedStoragePool resource is deleted, the corresponding Storage Pool remains in the backend 745
Restrictions on changing ReplicatedStorageClass spec 746
Deleting a child StorageClass when deleting a ReplicatedStorageClass 746
Errors when creating a Storage Pool or StorageClass 746
Error “You’re not allowed to change state of linstor cluster manually” 746
Restoring the database from a backup 746
Backup restoration procedure 746
Missing sds-replicated-volume service Pods on a selected node 748
Additional support 749
Migration from the linstor module to sds-replicated-volume 749
Migration procedure 749
Migrating to ReplicatedStorageClass 750
Migrating to ReplicatedStoragePool 751
Migration from the sds-drbd module to sds-replicated-volume 751
Migration procedure 751
Reasons to avoid RAID with sds-replicated-volume 752
Recommendations for using local disks 752
Manual trigger the certificate renewal process 752

- Additional resources 753
- Distributed Ceph storage 753
  - Migration from ceph-csi module 753
  - Connecting to a Ceph cluster 754
  - Additional information 755
    - How to get a list of RBD volumes separated by nodes 755
    - Which versions of Ceph clusters are supported 756
    - Which volume access modes are supported 756
    - Examples of permissions (caps) for Ceph users 756
      - RBD 756
      - CephFS 756
      - CephFS + RBD 756
  - Additional resources 757
- HPE data storage 757
  - System requirements 757
  - Configuration 757
    - Enabling the module 757
    - Creating a StorageClass 758
    - Module health verification 758
  - Additional resources 758
- Huawei data storage 759
  - System requirements 759
  - Configuration 759
    - Enabling the module 759
    - Creating a StorageClass 759
    - Module health verification 760
  - Additional resources 760
- NetApp data storage 761
  - System Requirements 761
  - Configuring cluster integration with the NetApp storage system 761
  - Additional resources 762
- NFS storage 762
  - System Requirements 762
    - Minimum Requirements 763
    - Configuration Recommendations 763
  - Limitations 763
    - General Limitations 763
    - Volume Snapshot Limitations 763
    - RPC-with-TLS Mode Limitations 763
      - Functional Limitations 763
      - System Requirements 763
  - Configuration 764
    - Enabling the Module 764
    - Creating StorageClass 764
    - Volume Cleanup Configuration 765
      - Parameter RandomFillSinglePass 765
      - Parameter RandomFillThreePass 765
      - Parameter Discard 765
  - Changing NFS Server Parameters for Created PersistentVolumes 765
  - Creating Volume Snapshots 765
  - Issues with Deleting PVs with RPC-with-TLS Support 766
    - Adding Multiple CAs to the tlsParameters.ca Parameter 766
  - Additional resources 766
- S3 object storage 767
  - System requirements 767
  - Configuration 767
    - Enabling the module 767
    - Creating a StorageClass 768
  - Module health check 768
  - Limitations 768
  - Known bugs 769
  - Changing connection parameters for existing PVs 769

- Displaying the mounted directory size 769
- Handling exceeded bucket or user quotas 769
- Obtaining information about used space 769
- Using multiple S3 storages in a single module 769
- Using a single bucket for multiple Pods 770
- Troubleshooting 770
  - Issues creating a PVC 770
  - Issues starting containers 770
- Additional resources 770

SCSI-based data storage 770
- Supported features 770
- Limitations 770
- System requirements 770
- Quick start 771
  - Enabling the csi-scsi-generic module 771
  - Creating an SCSITarget 771
    - iSCSI 771
    - Fibre Channel 772
  - Creating a StorageClass 773
  - Csi-scsi-generic module health check 774
- Additional resources 774

TATLIN.UNIFIED (Yadro) unified storage 774
- Enabling the module 774
- Connecting to the TATLIN.UNIFIED storage system 774
- Creating a StorageClass 775
- Ensuring the module works 775
- Additional resources 775

Virtualization 775
- Installation and configuration 776
- Storing images and disks 776
- Virtual machine classes 776
- Device passthrough 776
- Operating a cluster with virtual machines 776
- Additional resources 776

Virtual machine image storage 777
- Size and storage class 777
- Cleaning up image storage 777
- Additional resources 778

Storage for virtual machine disks and images 778
- Additional resources 778

Cluster images of virtual machines 779
- Image types and formats 779
- Creating a cluster image from an HTTP server 780
- Creating a cluster image from a container image registry 780
- Uploading a cluster image from the command line 781
- Additional resources 782

Virtual machine classes 783
- Default VirtualMachineClass 783
- VirtualMachineClass settings 784
- Additional resources 785

Virtual processor of virtual machines 785
- Virtual processor 785
- vCPU Discovery configuration example 786
- Additional resources 788

Placement of virtual machines across nodes 788
- Additional resources 789

Virtual machine sizing policy 789
- Additional resources 792

CPU oversubscription for virtual machines 793
- Fixed oversubscription 793
- Oversubscription chosen by the project owner 793

USB devices in virtual machines 794

- Path of a USB device from a node to a VM 794
- Discovered devices (NodeUSBDevice) 795
  - Assigning a namespace to a USB device 795
- Viewing USB device details 795
- Requirements and limitations 796
- Additional resources 796

GPU devices in virtual machines 796
- Additional resources 797

PCI devices in virtual machines 797
- Node requirements 797
- Assigning a namespace to a PCI device 798
- Requirements and limitations for PCI devices 798
- Additional resources 799

Security 799
- Additional resources 799

Security events 800
- Dependencies and requirements 800
- Data sources for security events 800
- Enabling security events 800
- Configuring event sources 801
- Configuring event delivery 802
  - Configuring a destination 802
  - Configuring event delivery rules 802
- Default module settings 803
- Supported security events 803
- Additional resources 804

Kubernetes API event audit 804
- Built-in audit policies 804
- Configuring a custom audit policy 805
- How to generate the contents of the audit-policy.yaml file 805
  - Examples 806
    - Log all requests from all authenticated users 806
    - Logging all requests not from service accounts 807
    - Logging all operations with standard resources 807
- Working with the audit log file 807
- Redirecting the audit log file to stdout 808
- Additional resources 808

Runtime audit 808
- Data sources for runtime audit 809
- Minimum requirements 809
- Enabling runtime audit 809
  - Manually configuring the Kubernetes API audit webhook 809
- Working with audit rules 810
  - Adding a custom rule 810
  - Applying a third-party rule 811
- Log collection and alerts 812
  - Configuring log and event collection 812
  - Configuring critical event alerts 812
  - Viewing metrics 813
- Debugging and simulating events 813
  - Enabling debug logging 813
  - Simulating events 814
    - Falco 814
    - Falcosidekick 814
- Additional resources 815

Virtualization event audit 815
- Enabling the audit 815
- Event types 815
- Viewing events 816
- Additional resources 816

Security policies 816
- How validation failure messages are displayed 817

- Pod validation when policies are modified or added 817
- Applying Pod Security Standards 818
  - Default policy 818
  - Assigning a policy 818
  - Enforcement modes 819
  - Policies in system namespaces 819
  - Extending a policy 820
- Operational policies 822
- Security policies 822
- Security policy exceptions 825
  - Adding exceptions 825
  - Configuration example 825
  - Partial policy enforcement 826
- Gatekeeper custom resources 828
- Custom Gatekeeper policy examples 829
  - Preventing deletion of a node without a specified label 829
  - Preventing exec and attach operations to specific Pods 830
    - Pods labeled heritage: deckhouse 830
    - Custom policy example 831
- Image signature verification 832
- Using alternative security policy management tools 834
- Additional resources 834

Container image vulnerability scanning 834
- Vulnerability scanning 834
- Scanning in namespaces 834
- Start conditions and scanning process 835
  - Forcing a rescan 835

Certificate management 835
- Monitoring 835
- Access roles 836
- Working with certificate issuers 836
  - Adding a ClusterIssuer with DNS-01 validation via webhook 836
  - Adding a ClusterIssuer that uses a custom certificate authority (CA) 836
  - Adding an Issuer and ClusterIssuer using HashiCorp Vault to request certificates 837
- Additional resources 838

Integration with software and security solutions 838
- Sending logs to KUMA 838
  - Example ClusterLogDestination and ClusterLoggingConfig configurations 839
- KCS configuration features 841
- Antivirus scanning exclusions for nodes 842
  - KESL configuration recommendations 842
- Additional resources 843

Automatic distribution of secrets across namespaces 843
- How it works 843
  - Incompatibility with multitenancy mode 843
- Configuring automatic secret distribution 843
- Distributing secrets to selected namespaces 844

Monitoring in Deckhouse Platform 844
- Hardware resource monitoring 844
- Kubernetes monitoring 844
- Ingress monitoring 845
  - Disabling collection of detailed statistics from Ingress resources 845
- Control plane monitoring 845
- Cluster monitoring 845
- Extended monitoring mode 845
- Alerts 846
  - Sending alerts to external systems 846
- Cluster SLA monitoring 846
- Additional resources 846

Configuring network interaction and cluster node monitoring 846
- Network interaction monitoring 847
  - Adding additional IP addresses for monitoring 847

Cluster node monitoring 847
Additional resources 847
Writing Prometheus data to longterm storage 848
Example of minimal PrometheusRemoteWrite 848
Example of advanced PrometheusRemoteWrite 848
Additional resources 848
External access configuration 849
Connecting Prometheus to Third-Party Grafana 849
Connecting Third-Party Applications to Prometheus 851
Metrics Collection via Gateway (Pushgateway) 852
Removing Metrics from Gateway (Pushgateway) 853
Additional resources 854
Configuring a system for collecting and storing metrics 854
Prometheus capabilities 854
Prometheus operation 854
Using the global.modules.storageClass parameter for Prometheus 855
Additional resources 855
Cluster SLA monitoring 855
Module configuration 855
Interface 855
Status metrics export 856
Authentication 857
Behavior of upmeter pods 857
Additional resources 858
Configuring integrations 858
Redirecting alerts to Zabbix 858
Installation 858
Troubleshooting 861
Sending alerts to Telegram 862
Example of sending alerts to Slack with filter 862
Example of sending alerts to Opsgenie 863
Example of sending alerts via email 864
Additional resources 865
Logging 865
Log collection and delivery 866
Configuring log collection and delivery 866
Integration with external systems 867
Elasticsearch 867
Using index templates 867
Support for Elasticsearch < 6.X 868
Splunk 868
Logstash 869
Graylog 869
Message formats 869
Syslog 870
CEF 870
Log transformation 870
Converting records to structured objects 871
Label replacement 871
Label removal 872
Log filtering 873
Collecting logs from a specific container 873
Collecting logs without a specific string 873
Kubelet audit events 874
DP system logs 874
Log buffering 875
Example of enabling in-memory buffering 875
Example of enabling disk buffering 875
Example of defining buffer overflow behavior 875
Debugging and advanced features 876
Enabling debug logs for the log-shipper agent 876
Additional information about log transmission channels 876

Topology overview 876
Monitoring channel load 877
Getting raw and intermediate logs 877
Debugging VRL rules 877
Adding support for new source or sink 877
Special cases 878
Collecting logs from production namespaces using the labelSelector option 878
Label for excluding pods and namespaces 878
Additional resources 878
Short-term log storage 879
Integration with Grafana Cloud 879
Migration from Grafana Promtail 879
Additional resources 880
Multitenancy 880
Additional namespaces 880
Additional resources 880
Project management 881
Project creation 882
Automatic project creation for a namespace 883
Creating a project without specifying a template 884
Project naming rules 884
Project status and diagnostics 885
Service objects of a project 885
Virtual projects 886
Additional project namespaces 886
What applies to the additional namespaces 887
Labels of the project namespaces 887
Standard project fields 888
Granting access within a project 889
Roles available in a RoleBinding inside a project 890
Structured templates 890
Template parametrization 891
Template checks 892
Creating a custom project template 893
Labels for the objects of a Helm text 893
Skipping the heritage label 893
Excluding objects from management 893
Implementing validation of object changes with a custom label 894
How validation works in multitenancy-manager 894
Creating your own validation 894
Managing access to cluster-wide resources 896
Additional resources 896
Overview 896
How it works 897
Managed PostgreSQL 897
Enabling the service 897
Dependencies for specific features 897
Example of creating a PostgresClass 898
Change and delete a PostgresClass 901
Limit CPU and memory resources 901
Manage fault tolerance across availability zones 902
Automatically validate PostgreSQL settings 903
Set default values and allow overrides 903
Automatic workMem calculation 903
Bind to dedicated nodes 904
Additional resources 905
Backup and restore 905
Manual cluster restore 905
Restoring a cluster with a single control plane node 905
Restoring a multi-master cluster 906
Restoring individual objects 907
Restoring Kubernetes objects from an etcd backup 907

Example of steps to restore objects from an etcd backup 907
Restoring cluster objects from exported YAML files 910
Restoring the licensing cluster key 910
Restoring control plane component manifests from a control-plane-manager backup 911
Restoring objects after changing the master node IP address 913
Automated object extraction when changing IP address 913
Manual object restore after changing the IP address 914
Creating backups with Deckhouse CLI 915
Backing up etcd 916
Automatic etcd backup 916
Cluster configuration backup 917
Exporting logs from Loki 918
Additional resources 919
Automatic assignment of namespace labels and annotations 920
How it works 920
Configuring automatic label and annotation assignment 920
Example configuration 920
Additional resources 920
Application delivery with Argo CD 921
Additional resources 921
Running Argo CD 921
Preparing to run Argo CD 922
Deploying an Argo CD instance 923
Deploying multiple Argo CD instances 924
Advanced settings 925
Enabling high availability mode 925
Granting access to cluster resources 925
Using a custom cluster domain 928
Additional resources 928
Configuring authentication and authorization 928
Local authentication 928
Creating additional local users 929
Creating a user token 929
Disabling local authentication 930
SSO authentication 930
Using a self-signed certificate 931
Creating a user token 932
Additional resources 932
Configuring role-based access control 933
Built-in roles 933
Default policy for authenticated users 934
Anonymous access 935
Role-based access control structure 935
Role binding 935
Access policy 935
Resources and supported actions 936
Application-bound policies 936
The applications resource 936
Detailed permissions for update and delete 936
The action action 937
The override action 937
Other resources 937
The applicationsets resource 938
The logs resource 938
The exec resource 938
The extensions resource 938
The deny effect 938
Policy evaluation order and matching modes 938
Matching in glob mode 938
Using SSO users and groups 939
Local users 939
Testing RBAC policies 940

- Additional resources 940
- Marketplace 940
  - Administrator tasks 940
  - Marketplace resources 940
  - Additional resources 941
- Package repositories 941
  - Authentication and scan interval 941
    - Authentication 941
    - Scan interval 942
  - Checking repository status 942
  - Viewing discovered package versions 943
  - Additional resources 943
- Scanning 943
  - Viewing scan operations 943
  - Inspecting a scan operation 944
  - Triggering a manual scan 944
    - Full scanning 945
  - Additional resources 945
- Configuring package access 945
  - Allowing packages in a project 945
  - Allowing packages in one namespace of a project 946
  - Denying packages 947
  - Packages in a project template or in a project 947
  - Checking the available packages 948
  - Applications installed before a restriction 948
  - Limitations 948
  - Additional resources 948
- Platform update 949
- Configuring updates 949
  - Checking the current release channel 949
  - Switching release channels 950
  - Update modes 950
    - Patch versions only within the current minor (AutoPatch) 950
    - Automatic updates for all versions (Auto) 951
    - Manual mode (Manual) 951
    - Checking the current update mode 951
    - How automatic updates are applied 951
      - Release pinning 951
    - Manual update approval 952
  - Update windows 953
    - Applying updates when update windows are configured 953
    - Configuring update windows 953
      - Configuration examples 953
  - Updating virtualization 954
  - Additional resources 954
- Configuring notifications on new releases 954
  - Alerts in the monitoring system 954
  - Configuring notifications 955
    - Supported notifications parameters 956
  - Notification format 957
  - Additional resources 958
- Updating Kubernetes and versioning 958
  - Updating and version management 958
    - Changing the Kubernetes version 958
  - Monitoring Kubernetes update progress 959
    - ConfigMap content examples 960
      - Cluster up to date (3 master nodes, 3 worker nodes) 960
      - Start of update (e.g., Kubernetes version downgrade) 961
      - Control plane update in progress 962
      - Intermediate step of a multi-hop upgrade (e.g. 1.33 → 1.35) 963
      - Cluster up to date (2 master nodes, 1 arbitr node and 3 worker nodes) 965
      - Failure of one or more control plane components 966

- Additional resources 967
- Switching between DP editions 967
  - Pre-switch preparation 968
    - Queue check 968
    - Determining the current edition and version 968
    - Checking whether switching to the desired edition is possible 968
  - Switching the edition 978
    - Selecting a switching method 978
    - Switching using the registry module 979
    - Switching without the registry module 983
  - Additional resources 994
- Uninstallation 994
  - Deleting a cluster deployed with a cloud provider 994
  - Deleting a hybrid cluster 995
  - Deleting a static cluster 996
  - Additional resources 997
- Going to production 997
  - Release channel and update mode 997
  - Kubernetes version 997
  - Resource requirements 998
  - Things to consider when configuring 998
    - Master nodes 998
    - Frontend nodes 998
    - Monitoring nodes 999
    - System nodes 999
    - Cluster subnets 999
    - Storage 1000
    - Virtualization 1000
      - Node requirements for VMs 1000
      - Number of VMs per node 1000
      - VM subnets 1000
      - Live migration network 1001
      - Running VMs on master nodes 1001
      - Storage for VM disks 1001
      - Image storage 1001
      - VM classes 1001
  - Access control 1001
  - Configuring alerts 1002
  - Collecting logs 1002
  - Backups 1002
  - Community 1003
  - Additional resources 1003
- Picking resources for a bare metal cluster 1003
  - Installation planning 1003
  - Deployment scenarios 1003
  - Deciding on the amount of resources needed for nodes 1004
    - Single-node cluster 1004
    - Configuration examples 1004
  - Node and network requirements 1005
  - Community 1005
  - Additional resources 1005
- Updating DP in air-gapped environment 1005
  - Platform update mechanism using release channels 1005
  - Module from source update mechanism 1008
  - Vulnerability scanner database update mechanism 1009
  - Example workflow for updating the platform, modules, and vulnerability databases 1009
  - Possible issues 1010
    - Release is suspended 1010
  - Additional resources 1011
- Disk layout and size 1011
  - Where and what does Deckhouse Platform store 1012
  - Disk space recommendations 1012

Master nodes 1012
Worker nodes 1013
System nodes 1013
Pod log storage 1013
Trivy vulnerability database repository 1013
Registry store on master nodes 1014
If resource limits are configured 1014
LVM-based local storage 1014
Additional resources 1014
Installing Deckhouse Platform in a private environment 1014
Private environment specifics 1015
Infrastructure selection 1015
Preparing a private container registry 1016
Installing a container registry 1016
Configuring Harbor 1027
Copying Deckhouse Platform images to a private container registry 1030
Signing in to the registry to run the installer 1033
Preparing VMs for the future nodes 1034
VM requirements 1034
Mapping registry.local to the Harbor VM 1034
Configuring SSH access to the master node 1035
Creating a user for the worker node 1035
Preparing the configuration file 1036
Installing Deckhouse Platform 1041
Adding nodes to the cluster 1042
Configuring the Ingress controller and creating a user 1043
Installing the ingress controller 1043
Creating a user to access the cluster web-interface 1044
Configuring DNS records 1045
Next steps 1046
Deploying your first application 1046
Learn more 1046
Additional resources 1046
Switching CNI in the cluster 1047
Method 1: Using the d8 network cni-migration command group of the d8 utility (automated switching) 1047
Starting the migration 1047
Monitoring progress 1047
Completion and cleanup 1048
Method 2: Using the d8 k commands (manual switching) 1048
Starting the migration 1048
Monitoring progress 1048
Completion and cleanup 1049
Troubleshooting 1049
Agent does not start on a node 1049
Node stuck in CleaningNodes phase 1049
Target CNI pods do not start 1049
Migration stuck 1049
Additional resources 1049
Planning for a high pod count on a node 1050
Pod limit per node 1050
Node kernel 1050
Node disk 1050
Application images 1051
Node resources 1051
Additional resources 1051
Installing Deckhouse Platform in an existing Talos cluster 1051
Prerequisites 1052
Setting the working paths 1052
Preparing an administrative kubeconfig 1052
If you already have a kubeconfig 1053
If you need to obtain a kubeconfig through Talos 1053

Verifying permissions 1053
Verifying the existing cluster 1053
Verifying component ownership 1054
Preparing a kubeconfig for the installer container 1055
Creating the configuration file 1055
Running the Deckhouse Platform Open installer 1056
Monitoring the installation 1056
Verifying the installation 1056
Verifying that DP does not manage Talos components 1057
Successful installation criteria 1057
Additional resources 1057
Migrating VMs from VMware to DP 1058
Migration approaches 1058
Direct VMDK import 1058
Recommended path 1058
Migration stages 1058
What you need for migration 1058
Install tools 1059
Convert the disk 1059
Extract an OVA 1059
Convert VMDK to qcow2 via virt-v2v 1060
Upload the disk to the cluster 1060
Choose a StorageClass 1060
Create a VirtualDisk for upload 1060
Get upload URLs 1061
Upload the image 1061
Check status 1061
Create the virtual machine 1062
VMware vs DP terminology 1062
VirtualMachine example 1062
Common spec fields 1063
Check VM status 1063
Connect to the VM 1063
Additional resources 1063
Architecture 1064
DP architecture 1064
Modules 1064
Additional resources 1065
DP architecture (C4 model) 1065
Legend 1065
Context diagram 1066
Subsystem diagram 1067
Module diagram 1068
Additional resources 1069
Deckhouse Platform module development 1069
Requirements 1070
Before you start 1070
Got a question? 1070
Module structure 1071
charts 1072
crds 1072
docs 1073
hooks 1074
images 1075
openapi 1075
conversions 1075
Conversion tests 1076
config-values.yaml 1076
Validations with x-deckhouse-validations 1076
Examples of rules 1077
Scalar and array value validation 1077
Validation of additionalProperties (map) 1078

values.yaml 1078
templates 1078
.helmignore 1079
Chart.yaml 1079
module.yaml 1079
Configuring module accessibility in DP editions 1081
Module accessibility logic 1082
Configuration examples 1082
Additional resources 1083
Build and publish 1083
Types of module artifacts 1083
Building module artifacts and publishing them to the container registry 1083
Additional resources 1084
How to run and verify a module in the DP cluster 1084
Run the module in the DP cluster 1084
Module source 1084
Switching the module to a different module source 1089
Module update policy 1089
Enabling the module 1090
Troubleshooting 1090
Integrating Deckhouse Module Tools for Module Validation 1091
For GitHub Projects 1091
For GitLab Projects 1092
Steps to connect 1092
Additional resources 1092
Dependencies of the Deckhouse Platform modules 1092
Deckhouse Platform version dependency 1093
Kubernetes version dependency 1093
Dependency on the version of other modules 1094
Required dependencies 1094
Conditional module requirements 1095
Restrictions on enabling and disabling prometheus when there is an conditional dependency on test 1095
Restrictions on updating prometheus when there is an conditional dependency on test 1095
Restrictions on enabling test, whose version is specified in prometheus dependencies 1096
Restrictions on updating test, whose version is specified in prometheus dependencies 1096
Versioning modules 1096
Release channels 1096
Procedure for publishing a new version 1097
Module lifecycle 1097
Definition of module stability 1097
API versioning 1097
Release of the new API version 1098
Automatic API Conversion 1098
Recommendations for Releasing New Versions of CRD 1098
Additional resources 1098
An example of creating a module with adaptation of an existing chart 1098
Preparing module source code and building 1098
Publishing the module to the release channel 1100
Enabling a module in a cluster 1101
Migration of ModuleUpdatePolicy to version v1alpha2 1103
Additional resources 1103
Module development and debugging 1103
Module availability and enabling by default 1105
How ModulePullOverride works 1105
An example 1106
Module auto-update logic 1108
Skipping intermediate releases (from-to) 1108
Examples 1110
Module artifacts in the container registry 1111
Print the list of modules in the module source 1112
Print the list of module images 1112

Print the list of files in the module image 1112
Print the list of images of the module’s application containers 1112
Configuring extra images 1112
Print the list of releases 1113
Print the version in use on the alpha release channel 1113
Additional resources 1113
Marketplace 1113
Sections 1114
Concepts 1114
Package types 1114
Resource model 1114
ApplicationPackageVersion content 1115
Application installation steps 1115
Application constraints 1115
Functional constraints 1116
Naming constraints 1116
Additional resources 1116
Application development 1116
Prerequisites 1116
Bootstrapping an Application package 1116
Project structure 1117
File package.yaml 1117
Requirements 1118
Deletion confirmation 1119
File changelog.yaml 1120
OpenAPI schemas 1120
Templates and hooks 1120
Local rendering 1120
Linting 1120
File .pkglint.yaml 1121
Local build 1122
CI/CD setup 1122
Environment variables 1122
Triggering a release 1122
Package artifacts in the container registry 1123
Bundle contents 1123
Version metadata image contents 1123
Release channels 1123
Additional resources 1124
Application settings 1124
Schema files 1124
How DP builds values 1124
Validation 1125
Undeclared fields 1125
When settings are validated 1125
CEL rules 1125
Defaulting a grantable cluster-wide resource value 1126
Immutable fields 1126
Web interface form 1127
Confirmation before switching a mode 1128
Multi-line and masked fields 1129
Memory size fields 1130
Additional resources 1130
Templates 1130
Template values 1131
Root .Application 1131
Root .Platform 1131
Object names 1131
Labels and object protection 1132
Container images 1132
Ingress and HTTPS 1133
Application endpoints 1134

HTTPRoute endpoints 1135
ServiceAccount endpoints 1136
Credentials in the web interface 1136
ServiceAccount token 1137
Restrictions 1137
Replica limits 1137
Workload health 1138
Additional resources 1138
Nelm annotations 1139
Deployment stages 1139
Render 1139
Plan 1139
Apply 1139
Common use cases 1139
Job that must run before the main application 1139
Preserve a resource across uninstall or chart removal 1140
Resource shared between releases 1140
Wait for a resource created by an operator 1140
Non-critical component that must not fail the deploy 1141
Resource rendered only on first install 1141
Slow-starting resource with a large image 1141
Ordering and dependencies 1142
Weights 1142
Direct dependencies 1142
External dependencies 1142
Delete dependencies 1142
Lifecycle annotations 1143
helm.sh/resource-policy 1143
werf.io/resource-policy 1143
werf.io/ownership 1143
werf.io/deploy-on 1143
werf.io/delete-policy 1144
werf.io/delete-propagation 1144
Tracking annotations 1144
Log annotations 1144
Template functions 1145
werf_secret_file 1145
dump_debug, printf_debug, include_debug, tpl_debug 1145
Known issues and troubleshooting 1145
Additional resources 1145
Hooks 1146
Hooks binary 1146
Hook example 1146
Bindings and execution order 1147
Error handling 1148
ApplicationHookInput 1148
Kubernetes bindings 1149
Object operations 1149
Object names 1150
Values 1150
Settings validation hook 1150
Restrictions 1151
Additional resources 1151
Lifecycle and debugging 1151
Admission checks 1151
Installation pipeline 1152
Events and reactions 1152
Suspension 1152
Deletion 1152
Orphan resources 1153
Resource restoration 1154
Deployment tracking 1154

- Maintenance mode 1154
- Debugging 1155
  - Application status 1155
  - State in DP 1155
  - Logs 1156
- Additional resources 1156

Disaster resilience 1156

Geo-distribution 1156
- Using multiple availability zones (Multi-AZ) 1156
  - Availability zone features 1157
  - Distributing nodes across zones 1157
- Using multiple regions (Multi-Region) 1157
  - Region features 1157
  - Distributing nodes across regions 1157
- Balancing incoming traffic 1157
- Storage organization 1157

Geo-reserving 1158
- Balancing incoming traffic in a multi-cluster 1158
  - Balancing traffic via active-active method 1158
  - Balancing traffic via active-standby method 1159
- Storage organization in a multi-cluster 1160

Updating 1160
- Releases 1160
- Release channels 1160
  - Switching to a more stable channel 1161
  - Switching to a less stable channel 1161
- Control plane updates 1161
  - Patch version updates 1161
  - Automatic minor version updates 1161
  - Manual minor version updates 1161
  - Process of control plane component updates 1162
- Retrieving the changelog 1162
  - Changelog contents 1162
  - Minor versions and zero patch releases 1162
- Checking dependencies before update 1162
- Additional resources 1163

Deckhouse subsystem 1163

Web interface modules 1163
- Console module 1163
  - Module architecture 1163
  - Module components 1164
  - Module interactions 1164
- Documentation module 1165
  - Module architecture 1165
  - Module components 1166
  - Module interactions 1166
- Deckhouse-tools module 1166
  - Module architecture 1166
  - Module components 1167
  - Module interactions 1167
- Additional resources 1167

Deckhouse module 1167
- Module architecture 1168
- Module components 1169
- Module interactions 1170
- Additional resources 1170

Queuing mechanism 1171
- Addon-operator queues 1171
- Marketplace queues 1172
- Additional resources 1173

Registry module 1173
- Module architecture 1174

- Module components 1177
- Module interactions 1178
- Additional resources 1178

DP cluster management 1179
- Commander module 1179
  - Commander module architecture 1179
  - Commander module components 1180
    - Core components 1180
    - Service components 1182
    - Optional components 1183
  - Module interactions 1183
- Commander-agent module 1184
  - Commander-agent module architecture 1184
  - Commander-agent module components 1184
  - Module interactions 1185
- Additional resources 1185

Delivery subsystem 1185

Operator-argo module 1185
- Module architecture 1186
- Module components 1189
- Module interactions 1192
- Additional resources 1193

Kubernetes & Scheduling subsystem 1193

Cluster control plane 1193

Kubelet 1194
- Static pod manifests 1194
- Kubelet interactions 1194
- Additional resources 1195

Cluster control plane management 1196
- Control-plane-manager module 1196
  - Module architecture 1196
  - Module components 1197
  - Module interactions 1198
- Cluster control plane monitoring 1198
- Additional resources 1199

Vertical Pod Autoscaler 1199
- VPA operating modes 1199
- VPA limitations 1199
- Module architecture 1200
- Module components 1201
- Module interactions 1202
- Additional resources 1202

Adaptive-resource-management module 1203
- Module architecture 1203
- Module components 1203
- Module interactions 1203
- Additional resources 1203

Descheduler module 1204
- Module architecture 1204
- Module components 1204
- Module interactions 1205
- Additional resources 1205

Pod-reloader module 1205
- Module architecture 1205
- Module components 1206
- Module interactions 1206
- Additional resources 1206

Cluster & Infrastructure subsystem 1206

Node management 1207
- Node types and addition mechanics 1207
- Automatic deployment, configuration, and update of Kubernetes nodes 1208
  - Kubernetes node deployment 1208

Maintaining node state 1208
Working with nodes in supported clouds 1208
Cloud node scaling 1208
Working with Static nodes 1209
Node grouping and group management 1209
What is the Instance resource 1209
When node reboot is required 1209
NodeGroup parameter effects on node updates and restarts 1209
Additional resources 1210
Bashible 1210
Bashible scripts and the bashible service 1210
Bashible-api-server 1211
Node-manager module 1211
Module architecture 1211
Additional resources 1212
CloudEphemeral node management 1212
Module architecture 1212
Module components 1212
Module interactions 1214
Architecture features specific to CloudEphemeral nodes 1214
Additional resources 1214
CloudPermanent node management 1214
Module architecture 1215
Module components 1215
Module interactions 1216
Architecture features specific to CloudPermanent nodes 1216
Additional resources 1216
CloudStatic node management 1217
Module architecture 1217
Module components 1217
Module interactions 1218
Architecture features specific to CloudStatic nodes 1219
Additional resources 1219
Static node management 1219
Module architecture 1219
Module components 1220
Module interactions 1221
Architecture features specific to Static nodes 1221
Additional resources 1221
Hybrid node group and cluster management 1222
Additional resources 1222
Cloud-provider-aws module 1222
Module architecture 1222
Module components 1223
Module interactions 1224
Additional resources 1225
Cloud-provider-azure module 1225
Module architecture 1225
Module components 1226
Module interactions 1226
Additional resources 1227
Cloud-provider-dvp module 1227
Module architecture 1227
Module components 1228
Module interactions 1229
Additional resources 1230
Cloud-provider-dynamix module 1230
Module architecture 1230
Module components 1231
Module interactions 1232
Additional resources 1233
Cloud-provider-gcp module 1233

Module architecture 1233
Module components 1234
Module interactions 1234
Additional resources 1235
Cloud-provider-huaweicloud module 1235
Module architecture 1235
Module components 1236
Module interactions 1237
Additional resources 1238
Cloud-provider-openstack module 1238
Module architecture 1238
Module components 1239
Module interactions 1240
Additional resources 1240
Cloud-provider-vcd module 1240
Module architecture 1241
Module components 1242
Module interactions 1243
Additional resources 1244
Cloud-provider-vsphere module 1244
Module architecture 1244
Module components 1245
Module interactions 1246
Additional resources 1246
Cloud-provider-yandex module 1246
Module architecture 1247
Module components 1248
Module interactions 1249
Additional resources 1250
Cloud-provider-zvirt module 1250
Module architecture 1250
Module components 1251
Module interactions 1252
Additional resources 1253
Chrony module 1253
Module architecture 1253
Module components 1254
Module interactions 1255
Additional resources 1255
Gpu module 1255
DRA mode 1255
Module architecture 1255
Module components 1256
Module interactions 1257
Device Plugin mode 1257
Module architecture 1257
Module components 1258
Module interactions 1259
Additional resources 1259
Registry-packages-proxy module 1259
Module architecture 1260
Module components 1260
Module interactions 1260
Additional resources 1261
Terraform-manager module 1261
Module architecture 1261
Module components 1262
Module interactions 1263
Additional resources 1263
IAM subsystem 1263
Authentication 1263
Connecting to Kubernetes API using a generated kubeconfig 1263

- Protection against credential brute-forcing in Dex 1264
- Authentication via DexAuthenticator 1264
- DP-specific Dex extensions 1264
- High availability mode 1264
- Additional resources 1264

Multitenancy 1265
- Internal logic 1265
  - Project creation 1265
  - Project isolation 1265
  - Managing access to cluster-wide resources 1265
- Module architecture 1266
- Module components 1266
- Module interactions 1266
- Additional resources 1267

User-authn module 1267
- Module architecture 1267
- Module components 1271
- Module interactions 1272
- Additional resources 1272

User-authz module 1272
- Module architecture 1273
- Module components 1273
- Module interactions 1274
- Additional resources 1275

Security subsystem 1275

Integrity control 1275
- Container image integrity control at the CRI level 1275
- Integrity control of user workloads at startup 1276
- Integrity control of running containers 1276
- Integrity protection of DP modules 1276
- Additional resources 1277

Runtime audit architecture 1277
- Audit rules 1277
  - Built-in rules 1277
  - Custom rules 1278
- Additional resources 1278

Security events architecture 1278
- Solution architecture 1278
  - Processing pipeline 1278
    - Collection layer (log-shipper) 1279
    - Processing and extraction 1279
    - Transformation and enrichment 1280
    - Filtering and delivery 1280
- Security event model 1280
- Minimal sufficient lifecycle 1281
- Supported security events 1281
- Additional resources 1281

Admission-policy-engine module 1281
- Module architecture 1282
- Module components 1283
- Module interactions 1284
- Additional resources 1284

Cert-manager module 1284
- Module architecture 1284
- Module components 1285
- Module interactions 1286
- Additional resources 1286

Module operator-trivy 1287
- Module architecture 1287
- Module components 1288
- Module interactions 1289
- Additional resources 1290

Runtime-audit-engine module 1290
Module architecture 1290
Module components 1291
Module interactions 1291
Additional resources 1292
Secrets-store-integration module 1292
Module architecture 1292
Module components 1293
Module interactions 1294
Additional resources 1294
Network subsystem 1295
Alb module 1295
Module architecture 1295
Module components 1296
Module interactions 1297
Additional resources 1297
Cni-cilium module 1298
Module architecture 1298
Module components 1299
Module interactions 1301
Additional resources 1301
Cni-flannel module 1301
Module architecture 1301
Module components 1302
Module interactions 1302
Additional resources 1303
Cni-simple-bridge module 1303
Module architecture 1303
Module components 1303
Module interactions 1304
Additional resources 1304
Cilium-hubble module 1304
Module architecture 1304
Module components 1305
Module interactions 1305
Additional resources 1306
Ingress-nginx module 1306
Module architecture 1306
Module components 1306
Module interactions 1307
Receiving traffic from external networks 1307
Architecture of Ingress controller with HostWithFailover inlet type 1308
Components of failover Ingress controller 1308
Failover Ingress controller interactions 1309
Additional resources 1309
Istio module 1309
Module architecture 1310
Module components 1312
Main module components 1312
Ambient mode components 1313
Federation or multicluster mode components 1313
Module interactions 1314
Additional resources 1315
Metallb module 1315
Module architecture 1315
Module components 1317
Module interactions 1317
Additional resources 1318
Sdn module 1318
Module architecture 1318
Module components 1319
Module interactions 1320

Additional resources 1320
Service-with-healthchecks module 1320
Module architecture 1321
Module components 1322
Module interactions 1323
Additional resources 1323
Static-routing-manager module 1323
Module architecture 1323
Module components 1324
Module interactions 1324
Additional resources 1324
DNS modules 1324
kube-dns module 1324
Module architecture 1324
Module components 1325
Module interactions 1325
node-local-dns module 1326
When using cni-cilium 1326
Module architecture 1326
Module components 1326
Module interactions 1327
When using cni-flannel or cni-simple-bridge 1327
Module architecture 1327
Module components 1328
Module interactions 1328
Additional resources 1328
Cluster-level Istio architecture 1328
Estimating overhead 1330
Control plane 1330
Data plane 1330
Additional resources 1330
Application service architecture with Istio enabled 1330
Architecture features 1330
User request lifecycle 1331
Application with Istio disabled 1331
Application with Istio enabled 1331
Additional resources 1331
Caching DNS server in a cluster 1331
Caching DNS server operation principles 1331
CoreDNS configuration aspects 1331
Grafana dashboard 1331
Additional resources 1332
Storage subsystem 1332
Local-path-provisioner module 1332
Module architecture 1332
Module components 1333
Module interactions 1333
Additional resources 1333
Snapshot-controller module 1334
Module architecture 1334
Module components 1335
Module interactions 1335
Additional resources 1336
Sds-local-volume module 1336
Module architecture 1336
Module components 1337
Module interactions 1338
Additional resources 1338
Sds-node-configurator module 1338
Module architecture 1338
Module components 1340
Module interactions 1340

Additional resources 1340
Csi-ceph module 1340
Module architecture 1340
Module components 1341
Module interactions 1342
Additional resources 1342
Csi-hpe module 1342
Module architecture 1342
Module components 1343
Module interactions 1344
Additional resources 1344
Csi-huawei module 1344
Module architecture 1344
Module components 1345
Module interactions 1345
Additional resources 1346
Csi-netapp module 1346
Module architecture 1346
Module components 1347
Module interactions 1347
Additional resources 1347
Csi-nfs module 1348
Module architecture 1348
Module components 1349
Module interactions 1350
Additional resources 1350
Csi-s3 module 1350
Module architecture 1350
Module components 1351
Module interactions 1351
Additional resources 1352
Csi-scsi-generic module 1352
Module architecture 1352
Module components 1353
Module interactions 1354
Additional resources 1354
Csi-vsphere module 1354
Module architecture 1354
Module components 1355
Module interactions 1355
Additional resources 1355
Csi-yadro-tatlin-unified module 1356
Module architecture 1356
Module components 1357
Module interactions 1358
Additional resources 1358
Standard CSI driver 1358
Driver architecture 1358
Driver components 1359
Driver interactions 1360
Additional resources 1360
Ceph RBD CSI driver 1361
Driver architecture 1361
Driver components 1361
Driver interactions 1362
Additional resources 1362
Huawei CSI driver 1363
Driver architecture 1363
Driver components 1364
Driver interactions 1364
Additional resources 1365
NFS CSI driver 1365

Driver architecture 1365
Driver components 1366
Driver interactions 1366
Additional resources 1367
S3 CSI driver 1367
Driver architecture 1367
Driver components 1367
Driver interactions 1368
SCSI Generic CSI driver 1368
Driver architecture 1368
Driver components 1369
Driver interactions 1370
VMware vSphere CSI driver 1370
Driver architecture 1370
Driver components 1371
Driver interactions 1372
Additional resources 1372
Observability subsystem 1373
Architecture of monitoring in the Deckhouse Platform 1373
Composition and interaction scheme of monitoring components 1373
Components installed by DP 1373
Prometheus 1374
What’s in the Prometheus pod? 1374
How is Prometheus configured? 1375
Where does Prometheus get the list of targets? 1375
How are Service Monitors processed? 1376
How are custom resources with rules processed? 1377
Architecture of DP component availability assessment (upmeter) 1377
Additional resources 1378
Prometheus module 1378
Module architecture 1378
Module components 1378
Module interactions 1380
Fault tolerance and High Availability (HA) monitoring mode 1380
Additional resources 1380
Observability module 1380
Module architecture 1381
Module components 1381
Module interactions 1382
Additional resources 1382
Extended-monitoring module 1383
Module architecture 1383
Module components 1384
Module interactions 1385
Additional resources 1385
Monitoring-kubernetes module 1385
Module architecture 1385
Module components 1386
Module interactions 1386
Additional resources 1387
Upmeter module 1387
Module architecture 1387
Module components 1387
Module interactions 1388
Additional resources 1388
Logging modules 1389
Log-shipper module 1389
Module architecture 1389
Module components 1389
Module interactions 1390
Loki module 1390
Module architecture 1390

Module components 1391
Module interactions 1391
Additional resources 1391
Managed Services subsystem 1391
Additional resources 1391
Managed-cassandra module 1391
Module architecture 1391
Module components 1392
Module interactions 1392
Additional resources 1392
Managed-clickhouse module 1393
Module architecture 1393
Module components 1393
Module interactions 1394
Additional resources 1394
Managed-hive-metastore module 1394
Module architecture 1394
Module components 1395
Module interactions 1395
Additional resources 1395
Managed-kafka module 1396
Module architecture 1396
Module components 1396
Module interactions 1397
Additional resources 1397
Managed-memcached module 1397
Module architecture 1397
Module components 1398
Module interactions 1398
Additional resources 1398
Managed-opensearch module 1398
Module architecture 1399
Module components 1399
Module interactions 1399
Additional resources 1400
Managed-postgres module 1400
Module architecture 1400
Module components 1401
Module interactions 1402
Additional resources 1403
Managed-starrocks module 1403
Module architecture 1403
Module components 1404
Module interactions 1404
Additional resources 1405
Managed-trino module 1405
Module architecture 1405
Module components 1405
Module interactions 1406
Additional resources 1406
Managed-valkey module 1406
Module architecture 1406
Module components 1407
Module interactions 1407
Additional resources 1408
Virtualization subsystem 1408
Additional resources 1408
Virtualization API 1408
Virtualization API architecture 1409
Virtualization API components 1409
Virtualization API interactions 1410
Additional resources 1410

The virtualization core 1410
Module core architecture 1411
Module core components 1412
Module core interactions 1413
Additional resources 1414
Deckhouse Virtualization Container Registry (DVCR) 1414
DVCR architecture 1414
DVCR components 1415
DVCR interactions 1415
Additional resources 1416
Importing and uploading VM images and disks 1416
Additional resources 1417
Auxiliary components 1417
Security events audit 1418
Architecture 1418
Components 1418
Interactions 1418
Virtualization-dra and other components 1418
Architecture 1418
Components 1419
Interactions 1420
Additional resources 1420
Global configuration 1420
Conversions 1421
Parameters 1421
Additional resources 1427
Custom Resources 1427
Application 1427
ApplicationPackage 1435
ApplicationPackageVersion 1436
ClusterConfiguration 1443
CNIMigration 1447
CNINodeMigration 1448
DeckhouseRelease 1449
InitConfiguration 1450
Module 1451
ModuleConfig 1457
ModulePullOverride 1458
ModuleRelease 1460
ModuleSettingsDefinition 1461
ModuleSource 1462
ModuleUpdatePolicy 1464
PackageRepository 1468
PackageRepositoryOperation 1470
SSHConfig 1473
SSHHost 1475
StaticClusterConfiguration 1475
Release channels 1476
Additional resources 1476
Supported Kubernetes and OS versions 1476
Linux 1476
Kubernetes 1477
Container registry 1477
Kubernetes conformance test results 1478
Test results 1478
CNCF Kubernetes conformance (Sonobuoy) 1478
1. Cluster and Deckhouse 1478
2. Adjust RBAC and admission (required for a clean run) 1478
3. Install Sonobuoy CLI 1478
4. Run conformance 1478
5. Fetch only e2e.log and junit_01.xml 1479
Additional resources 1479

Deckhouse Platform editions comparison 1479
Container registry structure 1479
Commercial editions container registry 1479
Network interaction of the platform components 1481
Requirements for network latency between master nodes 1481
Potential issues 1482
Checking current latencies 1482
Network latency at the other nodes 1482
Traffic between master nodes 1483
Traffic from master nodes to regular nodes 1483
Traffic from nodes to master nodes 1484
Traffic between nodes 1485
Static virtualization ports on cluster nodes (4100-4134/TCP reserved) 1492
External traffic to master nodes 1493
External traffic to frontend nodes 1493
External traffic for all nodes 1494
Additional resources 1494
The list of alerts 1495
Alert severity 1495
Module admission-policy-engine 1495
Module cert-manager 1498
Module chrony 1500
Module cloud-provider-aws 1501
Module cloud-provider-huaweicloud 1502
Module cloud-provider-openstack 1502
Module cloud-provider-yandex 1503
Module cloud-provider-zvirt 1506
Module cni-cilium 1506
Module cni-flannel 1510
Module cni-simple-bridge 1511
Module control-plane-manager 1513
Module deckhouse 1524
Module descheduler 1541
Module documentation 1541
Module istio 1543
Module kube-dns 1552
Module metallb 1552
Module multitenancy-manager 1553
Module node-local-dns 1562
Module node-manager 1563
Module okmeter 1577
Module openvpn 1577
Module registry 1579
Module secret-copier 1588
Module terraform-manager 1589
Module user-authn 1591
Module user-authz 1598
Module vertical-pod-autoscaler 1607
Sysctl parameters managed by the platform 1608
Additional resources 1610
Information about used software 1611
Used directors 1620
List of reserved Deckhouse Platform DNS names 1621
Hosts file data generator 1621
List of reserved DNS names 1621
Deckhouse CLI overview and installation 1625
Main command groups 1625
How do I install the Deckhouse CLI? 1625
trdl-based installation 1626
Installing the executable 1626
Deckhouse CLI command reference 1630
d8 backup 1630

d8 backup cluster-config 1630
d8 backup etcd 1630
d8 backup loki 1631
d8 completion 1631
d8 completion bash 1632
d8 completion fish 1632
d8 completion powershell 1632
d8 completion zsh 1633
d8 cr 1633
d8 cr catalog 1634
d8 cr config 1634
d8 cr digest 1634
d8 cr export 1635
d8 cr fs 1635
d8 cr fs cat 1636
d8 cr fs extract 1636
d8 cr fs ls 1637
d8 cr fs tree 1637
d8 cr login 1638
d8 cr ls 1639
d8 cr manifest 1639
d8 cr pull 1639
d8 cr push 1640
d8 data 1641
d8 data create 1641
d8 data delete 1641
d8 data download 1641
d8 data export 1642
d8 data export create 1642
d8 data export delete 1642
d8 data export download 1642
d8 data export list 1642
d8 data import 1643
d8 data import create 1643
d8 data import delete 1643
d8 data import upload 1643
d8 data list 1644
d8 delivery-kit 1644
d8 delivery-kit build 1644
d8 delivery-kit bundle 1649
d8 delivery-kit bundle apply 1649
d8 delivery-kit bundle copy 1655
d8 delivery-kit bundle plan 1656
d8 delivery-kit bundle publish 1661
d8 delivery-kit bundle render 1667
d8 delivery-kit chart 1672
d8 delivery-kit chart ts 1672
d8 delivery-kit chart ts build 1672
d8 delivery-kit chart ts init 1673
d8 delivery-kit ci-env 1675
d8 delivery-kit cleanup 1676
d8 delivery-kit completion 1681
d8 delivery-kit compose 1681
d8 delivery-kit compose config 1681
d8 delivery-kit compose down 1686
d8 delivery-kit compose run 1691
d8 delivery-kit compose up 1696
d8 delivery-kit config 1701
d8 delivery-kit config graph 1701
d8 delivery-kit config list 1702
d8 delivery-kit config render 1704
d8 delivery-kit converge 1705

d8 delivery-kit cr 1714
d8 delivery-kit cr login 1714
d8 delivery-kit cr logout 1715
d8 delivery-kit dismiss 1716
d8 delivery-kit export 1721
d8 delivery-kit helm 1725
d8 delivery-kit helm create 1726
d8 delivery-kit helm dependency 1727
d8 delivery-kit helm dependency build 1729
d8 delivery-kit helm dependency list 1730
d8 delivery-kit helm dependency update 1731
d8 delivery-kit helm env 1732
d8 delivery-kit helm get 1733
d8 delivery-kit helm get all 1734
d8 delivery-kit helm get hooks 1735
d8 delivery-kit helm get manifest 1736
d8 delivery-kit helm get metadata 1737
d8 delivery-kit helm get notes 1738
d8 delivery-kit helm get values 1739
d8 delivery-kit helm get-autogenerated-values 1740
d8 delivery-kit helm get-namespace 1744
d8 delivery-kit helm get-release 1746
d8 delivery-kit helm history 1747
d8 delivery-kit helm install 1749
d8 delivery-kit helm lint 1753
d8 delivery-kit helm list 1754
d8 delivery-kit helm package 1756
d8 delivery-kit helm plugin 1758
d8 delivery-kit helm plugin install 1759
d8 delivery-kit helm plugin list 1760
d8 delivery-kit helm plugin uninstall 1760
d8 delivery-kit helm plugin update 1761
d8 delivery-kit helm pull 1762
d8 delivery-kit helm registry 1764
d8 delivery-kit helm registry login 1765
d8 delivery-kit helm registry logout 1766
d8 delivery-kit helm repo 1767
d8 delivery-kit helm repo add 1768
d8 delivery-kit helm repo index 1769
d8 delivery-kit helm repo list 1770
d8 delivery-kit helm repo remove 1771
d8 delivery-kit helm repo update 1772
d8 delivery-kit helm rollback 1773
d8 delivery-kit helm search 1774
d8 delivery-kit helm search hub 1775
d8 delivery-kit helm search repo 1777
d8 delivery-kit helm secret 1778
d8 delivery-kit helm secret decrypt 1779
d8 delivery-kit helm secret encrypt 1780
d8 delivery-kit helm secret file 1782
d8 delivery-kit helm secret file decrypt 1782
d8 delivery-kit helm secret file edit 1784
d8 delivery-kit helm secret file encrypt 1785
d8 delivery-kit helm secret generate-secret-key 1786
d8 delivery-kit helm secret rotate-secret-key 1787
d8 delivery-kit helm secret values 1789
d8 delivery-kit helm secret values decrypt 1790
d8 delivery-kit helm secret values edit 1791
d8 delivery-kit helm secret values encrypt 1792
d8 delivery-kit helm show 1794
d8 delivery-kit helm show all 1794
d8 delivery-kit helm show chart 1796

d8 delivery-kit helm show crds 1797
d8 delivery-kit helm show readme 1799
d8 delivery-kit helm show values 1800
d8 delivery-kit helm status 1802
d8 delivery-kit helm template 1803
d8 delivery-kit helm test 1806
d8 delivery-kit helm uninstall 1807
d8 delivery-kit helm upgrade 1809
d8 delivery-kit helm verify 1812
d8 delivery-kit helm version 1813
d8 delivery-kit host 1814
d8 delivery-kit host cleanup 1815
d8 delivery-kit host purge 1817
d8 delivery-kit includes 1818
d8 delivery-kit includes get-file 1818
d8 delivery-kit includes ls-files 1820
d8 delivery-kit includes update 1821
d8 delivery-kit kube-run 1823
d8 delivery-kit lint 1827
d8 delivery-kit managed-images 1834
d8 delivery-kit managed-images add 1834
d8 delivery-kit managed-images ls 1837
d8 delivery-kit managed-images rm 1840
d8 delivery-kit plan 1843
d8 delivery-kit purge 1852
d8 delivery-kit render 1855
d8 delivery-kit rollback 1863
d8 delivery-kit run 1866
d8 delivery-kit sbom 1870
d8 delivery-kit sbom get 1871
d8 delivery-kit sbom merge 1876
d8 delivery-kit sbom validate 1877
d8 delivery-kit slugify 1878
d8 delivery-kit stages 1879
d8 delivery-kit stages copy 1879
d8 delivery-kit synchronization 1883
d8 delivery-kit verify 1884
d8 delivery-kit version 1886
d8 dist 1886
d8 dist check 1886
d8 dist plugins 1887
d8 dist plugins contract 1888
d8 dist plugins install 1888
d8 dist plugins list 1889
d8 dist plugins remove 1890
d8 dist plugins remove all 1891
d8 dist plugins versions 1892
d8 dist status 1893
d8 dist update 1893
d8 dist use 1894
d8 dist versions 1894
d8 help 1895
d8 iam 1895
d8 iam access 1895
d8 iam access grant 1895
d8 iam access revoke 1896
d8 iam get 1897
d8 iam get group 1897
d8 iam get rule 1897
d8 iam get user 1898
d8 iam group 1898
d8 iam group add-member 1899

d8 iam group create 1899
d8 iam group delete 1899
d8 iam group remove-member 1899
d8 iam list 1900
d8 iam list groups 1900
d8 iam list rules 1900
d8 iam list users 1901
d8 iam user 1901
d8 iam user create 1902
d8 iam user delete 1902
d8 iam user lock 1903
d8 iam user reset-password 1903
d8 iam user reset2fa 1904
d8 iam user unlock 1904
d8 k 1905
d8 k alpha 1906
d8 k alpha kuberc 1907
d8 k alpha kuberc set 1909
d8 k alpha kuberc view 1910
d8 k annotate 1912
d8 k api-resources 1914
d8 k api-versions 1916
d8 k apply 1917
d8 k apply edit-last-applied 1920
d8 k apply set-last-applied 1922
d8 k apply view-last-applied 1923
d8 k attach 1925
d8 k auth 1927
d8 k auth can-i 1928
d8 k auth reconcile 1930
d8 k auth whoami 1931
d8 k autoscale 1933
d8 k certificate 1935
d8 k certificate approve 1937
d8 k certificate deny 1938
d8 k cluster-info 1940
d8 k cluster-info dump 1941
d8 k completion 1943
d8 k config 1945
d8 k config current-context 1946
d8 k config delete-cluster 1947
d8 k config delete-context 1949
d8 k config delete-user 1950
d8 k config get-clusters 1951
d8 k config get-contexts 1953
d8 k config get-users 1954
d8 k config rename-context 1955
d8 k config set 1957
d8 k config set-cluster 1958
d8 k config set-context 1960
d8 k config set-credentials 1961
d8 k config unset 1963
d8 k config use-context 1964
d8 k config view 1966
d8 k cordon 1967
d8 k cp 1969
d8 k create 1970
d8 k create clusterrole 1972
d8 k create clusterrolebinding 1974
d8 k create configmap 1976
d8 k create cronjob 1978
d8 k create deployment 1980

d8 k create ingress 1982
d8 k create job 1984
d8 k create namespace 1986
d8 k create poddisruptionbudget 1988
d8 k create priorityclass 1990
d8 k create quota 1992
d8 k create role 1994
d8 k create rolebinding 1995
d8 k create secret 1997
d8 k create secret docker-registry 1999
d8 k create secret generic 2001
d8 k create secret tls 2003
d8 k create service 2005
d8 k create service clusterip 2006
d8 k create service externalname 2008
d8 k create service loadbalancer 2010
d8 k create service nodeport 2012
d8 k create serviceaccount 2014
d8 k create token 2016
d8 k debug 2017
d8 k delete 2020
d8 k describe 2022
d8 k diff 2024
d8 k drain 2026
d8 k edit 2028
d8 k events 2030
d8 k exec 2032
d8 k explain 2034
d8 k expose 2035
d8 k get 2038
d8 k kustomize 2040
d8 k label 2042
d8 k logs 2045
d8 k options 2047
d8 k patch 2048
d8 k plugin 2050
d8 k plugin list 2051
d8 k port-forward 2053
d8 k proxy 2054
d8 k replace 2056
d8 k rollout 2058
d8 k rollout history 2060
d8 k rollout pause 2062
d8 k rollout restart 2063
d8 k rollout resume 2065
d8 k rollout status 2067
d8 k rollout undo 2069
d8 k run 2070
d8 k scale 2073
d8 k set 2075
d8 k set env 2077
d8 k set image 2079
d8 k set resources 2081
d8 k set selector 2083
d8 k set serviceaccount 2085
d8 k set subject 2087
d8 k taint 2089
d8 k top 2091
d8 k top node 2093
d8 k top pod 2094
d8 k uncordon 2096
d8 k version 2097

d8 k wait 2099
d8 mirror 2101
d8 mirror pull 2101
d8 mirror push 2104
d8 network 2105
d8 network cni-migration 2105
d8 network cni-migration cleanup 2105
d8 network cni-migration switch 2106
d8 network cni-migration watch 2106
d8 network istio 2106
d8 package 2106
d8 package bootstrap 2106
d8 package bootstrap app 2107
d8 package bootstrap module 2107
d8 package build 2107
d8 package doc 2108
d8 package render 2109
d8 package verify 2109
d8 package verify remote 2110
d8 snapshot 2110
d8 snapshot create 2110
d8 snapshot delete 2111
d8 snapshot describe 2111
d8 snapshot download 2112
d8 snapshot get 2113
d8 snapshot local 2113
d8 snapshot local describe 2113
d8 snapshot local get 2113
d8 snapshot restore 2114
d8 snapshot upload 2115
d8 status 2117
d8 stronghold 2117
d8 stronghold audit 2117
d8 stronghold audit disable 2117
d8 stronghold audit enable 2117
d8 stronghold audit list 2117
d8 stronghold auth 2118
d8 stronghold auth disable 2118
d8 stronghold auth enable 2118
d8 stronghold auth help 2118
d8 stronghold auth list 2118
d8 stronghold auth move 2118
d8 stronghold auth tune 2118
d8 stronghold debug 2118
d8 stronghold delete 2118
d8 stronghold events 2119
d8 stronghold events subscribe 2119
d8 stronghold kv 2119
d8 stronghold kv delete 2119
d8 stronghold kv destroy 2119
d8 stronghold kv enable-versioning 2119
d8 stronghold kv get 2119
d8 stronghold kv list 2119
d8 stronghold kv metadata 2119
d8 stronghold kv metadata delete 2119
d8 stronghold kv metadata get 2120
d8 stronghold kv metadata patch 2120
d8 stronghold kv metadata put 2120
d8 stronghold kv patch 2120
d8 stronghold kv put 2120
d8 stronghold kv rollback 2120
d8 stronghold kv undelete 2120

d8 stronghold lease 2120
d8 stronghold lease lookup 2120
d8 stronghold lease renew 2121
d8 stronghold lease revoke 2121
d8 stronghold list 2121
d8 stronghold login 2121
d8 stronghold monitor 2121
d8 stronghold namespace 2121
d8 stronghold namespace create 2121
d8 stronghold namespace delete 2121
d8 stronghold namespace list 2121
d8 stronghold namespace lock 2122
d8 stronghold namespace lookup 2122
d8 stronghold namespace patch 2122
d8 stronghold namespace unlock 2122
d8 stronghold operator 2122
d8 stronghold operator generate-root 2122
d8 stronghold operator init 2122
d8 stronghold operator key-status 2122
d8 stronghold operator members 2122
d8 stronghold operator raft 2123
d8 stronghold operator raft autopilot 2123
d8 stronghold operator raft autopilot get-config 2123
d8 stronghold operator raft autopilot set-config 2123
d8 stronghold operator raft autopilot state 2123
d8 stronghold operator raft demote 2123
d8 stronghold operator raft join 2123
d8 stronghold operator raft list-peers 2123
d8 stronghold operator raft promote 2123
d8 stronghold operator raft remove-peer 2124
d8 stronghold operator raft snapshot 2124
d8 stronghold operator raft snapshot inspect 2124
d8 stronghold operator raft snapshot restore 2124
d8 stronghold operator raft snapshot save 2124
d8 stronghold operator rekey 2124
d8 stronghold operator rotate 2124
d8 stronghold operator seal 2124
d8 stronghold operator step-down 2124
d8 stronghold operator unseal 2125
d8 stronghold operator usage 2125
d8 stronghold patch 2125
d8 stronghold path-help 2125
d8 stronghold pki 2125
d8 stronghold pki health-check 2125
d8 stronghold pki issue 2125
d8 stronghold pki list-intermediates 2125
d8 stronghold pki reissue 2125
d8 stronghold pki verify-sign 2125
d8 stronghold plugin 2126
d8 stronghold plugin deregister 2126
d8 stronghold plugin info 2126
d8 stronghold plugin list 2126
d8 stronghold plugin register 2126
d8 stronghold plugin reload 2126
d8 stronghold plugin reload-status 2126
d8 stronghold policy 2126
d8 stronghold policy delete 2126
d8 stronghold policy list 2127
d8 stronghold policy read 2127
d8 stronghold policy write 2127
d8 stronghold print 2127
d8 stronghold print token 2127

d8 stronghold read 2127
d8 stronghold secrets 2127
d8 stronghold secrets disable 2127
d8 stronghold secrets enable 2127
d8 stronghold secrets list 2128
d8 stronghold secrets move 2128
d8 stronghold secrets tune 2128
d8 stronghold ssh 2128
d8 stronghold status 2128
d8 stronghold token 2128
d8 stronghold token capabilities 2128
d8 stronghold token create 2128
d8 stronghold token lookup 2128
d8 stronghold token renew 2129
d8 stronghold token revoke 2129
d8 stronghold transit 2129
d8 stronghold transit import 2129
d8 stronghold transit import-version 2129
d8 stronghold unwrap 2129
d8 stronghold version 2129
d8 stronghold version-history 2129
d8 stronghold write 2129
d8 system 2130
d8 system collect-debug-info 2130
d8 system collect-debug-info virtualization 2130
d8 system edit 2131
d8 system edit cluster-configuration 2131
d8 system edit provider-cluster-configuration 2132
d8 system edit static-cluster-configuration 2132
d8 system get 2132
d8 system get cluster-configuration 2133
d8 system get provider-cluster-configuration 2133
d8 system get static-cluster-configuration 2133
d8 system logs 2133
d8 system module 2134
d8 system module apply-now 2134
d8 system module approve 2135
d8 system module disable 2135
d8 system module enable 2135
d8 system module list 2136
d8 system module maintenance 2136
d8 system module maintenance disable 2136
d8 system module maintenance enable 2137
d8 system module snapshots 2137
d8 system module values 2137
d8 system package 2138
d8 system package scan 2138
d8 system queue 2138
d8 system queue list 2139
d8 system queue main 2139
d8 tools 2139
d8 tools cosign 2140
d8 tools far-convert 2140
d8 tools gostsum 2140
d8 tools htpasswd 2140
d8 tools imagedigest 2141
d8 tools imagedigest add 2142
d8 tools imagedigest calculate 2142
d8 tools imagedigest calculate-from-file 2142
d8 tools imagedigest validate 2143
d8 tools pki 2143
d8 tools pki certs 2143

d8 tools pki certs check 2143
d8 tools pki certs renew 2144
d8 tools pki certs renew all 2144
d8 tools sig-migrate 2144
d8 v 2145
d8 v ansible-inventory 2146
d8 v collect-debug-info 2148
d8 v console 2149
d8 v evict 2150
d8 v migrate 2151
d8 v port-forward 2152
d8 v restart 2153
d8 v scp 2155
d8 v spice 2156
d8 v ssh 2157
d8 v start 2159
d8 v stop 2160
d8 v vnc 2161
Glossary 2164
Active-active balancing 2164
Active-standby balancing 2164
Additional network 2164
Additional project namespace 2164
Addon-operator queue 2164
Admission policy 2165
Advanced load balancer 2165
Air-gapped cluster 2165
ALB (application load balancing) 2165
Alert 2165
Alert notification channel 2166
Alert severity 2166
Application endpoints 2166
Application instance 2166
Application reconciliation run 2166
Application suspension 2166
Arbiter node 2167
Automatic certificate issuance 2167
Availability probe 2167
Availability zone 2167
bashible 2168
Bashible bundle 2168
bashible-api-server 2168
Bastion host 2168
Bootstrap script 2168
bpfLB 2168
Built-in module 2169
Caching DNS server 2169
Canary deployment 2169
Chaos engineering 2169
CIS compliance check 2169
Cloud cluster 2169
Cloud provider 2170
CloudEphemeral node 2170
CloudPermanent node 2170
CloudStatic node 2170
Cluster 2170
Cluster API Provider Static (CAPS) 2170
Cluster configuration (ClusterConfiguration) 2170
Cluster managed by Deckhouse Platform 2171
Cluster network 2171
ColdStandby 2171
Conditional module dependency 2171

Consistent snapshot 2171
Control plane 2172
Core fraction 2172
CPU oversubscription 2172
Critical module 2172
CSI driver 2172
Dashboard 2172
Deckhouse CLI (d8) 2173
Deckhouse controller 2173
Deckhouse Platform API 2173
Deckhouse Platform component registry 2173
Deckhouse Platform image bundle 2173
Deckhouse Platform release 2174
Deckhouse web UI 2174
Deletion confirmation 2174
Deployment option 2174
Device passthrough 2174
Disaster resilience 2174
Disruptive update 2175
DNS name template 2175
DSR (Direct Server Return) 2175
DVCR 2175
Edition 2175
Egress gateway 2176
End-to-end authentication 2176
Extended monitoring 2176
External storage 2176
Federation 2176
Fencing 2176
Frontend node 2177
Geo-distribution 2177
Geo-reserving 2177
Global settings 2177
Grantable cluster-wide resource 2177
Guest OS agent 2178
HA mode 2178
High-level role 2178
Hook binding 2178
Host firewall 2178
Hybrid cluster 2179
Hybrid node group 2179
Image signature verification 2179
In-cluster image cache 2179
Infrastructure node 2179
Ingress controller 2179
Inlet 2180
Installation configuration file 2180
Installation into an existing cluster 2180
Installer 2180
Instance class 2180
Integrity control 2181
IPAM 2181
Istio Ingress Gateway 2181
Kubernetes API audit 2181
Kubernetes conformance 2182
kubernetes-api-proxy 2182
Layout 2182
License key 2182
Live migration 2182
Local storage 2182
Log buffering 2182
Log pipeline 2183

Long-term metrics storage 2183
Main cluster network 2183
Main project namespace 2183
Maintenance mode of a module or application 2183
Manage role 2184
Managed cluster 2184
Managed Gateway 2184
Managed service 2184
Managing cluster 2184
Marketplace 2184
Marketplace package 2185
Master node 2185
Minor version 2185
Module 2185
Module accessibility 2185
Module artifacts 2185
Module bundle 2186
Module dependencies 2186
Module exclusive group 2186
Module from a source 2186
Module hook 2186
Module lifecycle stage 2186
Module settings conversion 2187
Module source 2187
Module update policy 2187
Module weight 2187
Monitoring node 2187
Multi-master cluster 2188
Multicluster 2188
Namespace 2188
Network policy 2188
NLB (network load balancing) 2188
Node group 2188
Node group autoscaling 2189
Node maintenance mode 2189
Orphan resources of an application 2189
Package bundle 2189
Patch version 2189
Pod limit per node 2189
Pod redistribution 2190
Pod Security Standards 2190
Post-bootstrap script 2190
Preflight checks 2190
Priority class 2190
Private environment 2190
Project 2191
Project features 2191
Project namespace features 2191
Project network 2191
Protected parameters 2191
Quorum 2191
Region 2192
Registry management mode 2192
Release channel 2192
Release channel image 2192
Replicated storage 2192
Role model 2193
Runtime audit 2193
Runtime audit rule 2193
Runtime profile 2193
Scheduler extender 2193
Secret distribution 2193

- Secrets store 2194
- Security event 2194
- Service mesh 2194
- Short-term log storage 2194
- Single-master cluster 2194
- Single-node cluster 2195
- Sizing policy 2195
- Skipping intermediate module releases 2195
- SLA monitoring 2195
- Software-defined network (SDN) 2195
- Software-defined storage 2195
- Standby node 2196
- Static cluster 2196
- Static node 2196
- Static pod 2196
- Static user 2196
- Stronghold lease 2196
- Stronghold sealing 2197
- Stronghold secrets engine 2197
- Subsystem 2197
- Suspended release 2197
- System 2197
- System component 2197
- System namespace 2197
- System network 2198
- System node 2198
- The deckhouse system user 2198
- Thick volume 2198
- Thin volume 2198
- Underlay network 2198
- Update mode 2199
- Update window 2199
- Use role 2199
- User namespace 2199
- Virtual machine (VM) 2199
- Virtual machine block device 2199
- Virtual machine clone 2200
- Virtual machine disk 2200
- Virtual machine image 2200
- Virtual machine lifecycle 2200
- Virtual machine pool 2201
- Virtual machine snapshot 2201
- VM CPU topology 2201
- VM disk data export 2201
- VM eviction 2201
- VM migration policy 2202
- VM placement rules 2202
- VM pool disk reclaim policy 2202
- VM provisioning 2202
- VM rebalancing 2202
- VM run policy 2203
- Vulnerability database 2203
- Vulnerability scanning 2203
- Worker node 2203
- Workload 2203

Frequently Asked Questions 2203

- Common 2203
  - How can I apply an update for a specific module immediately? 2204
  - How can I apply an update DP immediately, bypassing update windows, canary releases, and manual update mode? 2204
  - What to do if, with VPN enabled, the container with the installer cannot access the network? 2204

How do I change the release channel for a module? 2205
How do I configure an external module to update to patch versions only? 2206
What should I do if the module image did not download and the module did not reinstall? 2207
Cluster and Infrastructure 2208
What to do if you encounter problems updating Kubernetes components on cluster nodes, synchronizing nodes, or applying NodeGroup Configuration? 2208
How to migrate node groups to Cluster API (CAPI)? 2209
What to do if problems adding a node to the cluster via Cluster API Provider Static are present? 2210
How do I replace the OS image in a cloud cluster? 2210
Requirements for the new image 2212
Changing the image in the configuration 2213
Applying the changes 2215
How to change the instance type for nodes with the CloudPermanent type? 2216
How to migrate a cloud provider to ModuleConfig-based configuration? 2218
How to migrate 2218
How to delete the legacy secret 2219
What to do if it takes a long time to switch to custom nodes in lower-priority groups? 2220
If you encounter issues with node ordering in a cluster with a single node group 2221
Example of the node ordering process with default settings 2221
What to do if the resources generated by an object are not created after adding the object? 2222
Deckhouse 2223
How can I check DP versions available in different release channels? 2223
How to check the job queue in Deckhouse? 2223
How to check the status of all Deckhouse task queues? 2223
How to view the status of the main task queue? 2224
What should I do if DP is not receiving updates from the configured channel? 2224
Example DNS settings that may cause issues resolving the DP image registry IP address 2224
Does Deckhouse support realtime (rt) and low-latency Linux kernels? 2225
How can I know when a new DP version is available for the cluster? 2225
How can I receive information about upcoming updates in advance? 2225
How can I tell that the update was successful? 2225
If something goes wrong 2226
How can I tell that an update is in progress? 2226
What to do if there are problems updating DP? 2226
Deckhouse Platform update fails, one or more Deckhouse pods are in an unworkable state 2226
DP update is stuck in the Release is suspended status 2227
IAM 2227
What to do if there are problems applying DexProvider settings? 2227
Kubernetes and Scheduling 2228
What to do when the API server is overloaded? 2228
How to check the Kubernetes version in use? 2228
Network 2229
Is the ingress-nginx module supported? Will it affect passing a PCI DSS audit? 2229
Virtualization 2229
How to use Ansible to provision virtual machines? 2229
How to automatically generate inventory for Ansible? 2230
How to use cloud-init to configure virtual machines? 2230
Updating and installing packages 2230
Creating a user 2231
Creating a file with required permissions 2231
Configuring disk and filesystem 2231
Configuring network interfaces for additional networks 2232
How to increase the DVCR size? 2233
How to change the DVCR StorageClass when a PVC already exists? 2234
How to install an operating system in a virtual machine from an ISO image? 2235
How to create a golden image for Linux? 2237
How to restore the cluster if images from registry.deckhouse.io cannot be pulled after a license change? 2239
How to redirect traffic to a virtual machine? 2240
Why is only one connection method available for a virtual machine? 2241
Why does VNC not work when the serial console is available? 2241

Why does the serial console not work when VNC is available? 2241
How to provide a Windows answer file (Sysprep)? 2242
Release notes for Deckhouse Platform 2246
Version 1.78 2246
Important 2246
Version 1.77 2246
Important 2246
Deckhouse subsystem 2247
Kubernetes & Scheduling subsystem 2247
IAM subsystem 2248
Security subsystem 2249
Cluster & Infrastructure subsystem 2249
Network subsystem 2250
1.77.10 2251
1.77.9 2251
1.77.8 2252
1.77.7 2252
1.77.6 2254
1.77.5 2256
1.77.4 2258
1.77.3 2259
1.77.2 2263
1.77.1 2266
Version 1.76 2268
Important 2268
Deckhouse subsystem 2268
Kubernetes & Scheduling subsystem 2269
IAM subsystem 2269
Security subsystem 2270
Cluster & Infrastructure subsystem 2270
Network subsystem 2271
1.76.18 2271
1.76.17 2272
1.76.16 2273
1.76.15 2273
1.76.14 2274
1.76.13 2275
1.76.12 2277
1.76.11 2277
1.76.10 2280
1.76.9 2282
1.76.8 2282
1.76.7 2283
1.76.6 2283
1.76.5 2284
1.76.4 2284
1.76.3 2285
1.76.2 2285
1.76.1 2286
Version 1.75 2286
Important 2286
Deckhouse subsystem 2287
Kubernetes & Scheduling subsystem 2287
IAM subsystem 2288
Security subsystem 2288
Cluster & Infrastructure subsystem 2288
Network subsystem 2289
1.75.16 2289
1.75.15 2289
1.75.14 2290
1.75.13 2290
1.75.12 2290

1.75.11 2290
1.75.10 2291
1.75.9 2291
1.75.8 2291
1.75.7 2292
1.75.6 2292
1.75.5 2292
1.75.4 2292
1.75.3 2294
1.75.2 2294
1.75.1 2294
Version 1.74 2295
Important 2295
Major changes 2295
Security 2295
Network 2295
1.74.18 2296
1.74.17 2296
1.74.16 2296
1.74.15 2297
1.74.14 2297
1.74.13 2297
1.74.12 2297
1.74.11 2298
1.74.10 2298
1.74.9 2298
1.74.8 2298
1.74.7 2299
1.74.6 2299
1.74.5 2299
1.74.4 2299
1.74.3 2300
1.74.2 2300
1.74.1 2301
Version 1.73 2301
Important 2301
Major changes 2302
Security 2302
Network 2303
1.73.14 2303
1.73.13 2304
1.73.12 2304
1.73.11 2304
1.73.10 2304
1.73.9 2304
1.73.8 2304
1.73.7 2305
1.73.6 2305
1.73.5 2305
1.73.4 2306
1.73.3 2306
1.73.2 2306
1.73.1 2306
Version 1.72 2307
Important 2307
Major changes 2307
Security 2307
1.72.13 2308
1.72.12 2308
1.72.11 2308
1.72.10 2308
1.72.9 2308

1.72.8 2308
1.72.7 2308
1.72.6 2309
1.72.5 2309
1.72.4 2309
1.72.3 2309
1.72.2 2309
1.72.1 2309
Version 1.71 2310
Important 2310
Major changes 2310
Security 2311
Network 2311
1.71.10 2312
1.71.9 2312
1.71.8 2312
1.71.7 2312
1.71.6 2312
1.71.5 2312
1.71.4 2313
1.71.3 2313
1.71.2 2313
1.71.1 2314
Version 1.70 2314
Important 2314
Major changes 2314
Security 2315
1.70.19 2315
1.70.18 2316
1.70.17 2316
1.70.16 2316
1.70.15 2316
1.70.14 2316
1.70.13 2316
1.70.12 2316
1.70.11 2316
1.70.10 2317
1.70.9 2317
1.70.8 2317
1.70.7 2317
1.70.6 2317
1.70.5 2317
1.70.4 2318
1.70.3 2318
1.70.2 2318
1.70.1 2318
Version 1.69 2318
Important 2318
Major changes 2319
Security 2319
1.69.17 2319
1.69.16 2320
1.69.15 2320
1.69.14 2320
1.69.13 2320
1.69.12 2320
1.69.11 2320
1.69.10 2320
1.69.9 2321
1.69.8 2321
1.69.7 2321
1.69.6 2321

1.69.5 2321
1.69.4 2321
1.69.3 2321
1.69.2 2322
1.69.1 2322
Version 1.68 2322
Important 2322
Major changes 2322
Security 2323
Mandatory component restart 2324
1.68.15 2324
1.68.14 2324
1.68.13 2324
1.68.12 2325
1.68.11 2325
1.68.10 2325
1.68.9 2325
1.68.8 2325
1.68.7 2325
1.68.6 2325
1.68.5 2326
1.68.4 2326
1.68.3 2326
1.68.2 2326
1.68.1 2326
Embedded Deckhouse Platform modules 2327
The admission-policy-engine module 2328
How validation failure messages are displayed 2328
Controller-level validation 2328
labelSelector semantics for controllers 2329
SecurityPolicyException label resolution for controllers 2329
Configuring controller-level validation 2329
Lenient mode for fields that may be filled in before the pod is created 2330
Constraints with limited controller-level coverage 2331
Pod validation when policies are modified or new ones are added 2331
Pod Security Standards 2332
Rules of the standards 2333
Operational policies 2333
Security policies 2334
Security policy exceptions 2336
Adding exceptions 2336
Configuration example 2336
Modifying Kubernetes resources 2337
Availability of the module components 2337
Why the validating webhook is a critical component 2337
Which objects are excluded from validation 2338
Third-party components 2338
Configuration 2338
Parameters 2340
Custom resources 2343
OperationPolicy 2343
SecurityPolicyException 2349
SecurityPolicy 2355
Custom Resources (by Gatekeeper) 2364
Mutation Custom Resources 2365
AssignMetadata 2365
Assign 2365
ModifySet 2366
AssignImage 2366
FAQ 2367
How do I configure alternative security policy management solutions? 2367
How do I configure policy selectors? 2367

When to use labelSelector 2368
Can matchNames and labelSelector be used together? 2368
Typical scenarios 2368
How do I extend Pod Security Standards policies? 2370
How to allow some Pod Security Standards policies without disabling whole list? 2371
What if there are multiple policies (operational or security) that are applied to the same object? 2373
Why is an Assign or AssignMetadata mutation rejected or silently ignored? 2374
How to apply a mutation only when a resource is created? 2375
Verification of image signatures 2375
How to block deleting a node without a label 2375
How to deny kubectl exec and kubectl attach to specific Pods? 2376
Built-in policy for finalizers 2377
Custom policy example 2377
How do I restrict GPU resource usage in namespaces? 2378
What to do if the validating webhook is unavailable? 2379
The cert-manager module 2379
Features of the cert-manager module (with the changes made) 2380
Restricting ClusterIssuers for projects 2380
Monitoring 2380
Module roles 2380
Third-party components 2380
Configuration 2380
Parameters 2382
Custom Resources 2389
Usage 2389
An example of provisioning a certificate 2389
Issuing a DNS wildcard certificate using Cloudflare 2389
Issuing a DNS wildcard certificate using Route53 2390
Issuing a DNS wildcard certificate using Google 2391
Issuing a DNS wildcard certificate using Yandex Cloud DNS 2392
Issuing a self-signed certificate 2393
FAQ 2393
What types of certificates are supported? 2393
How to add an additional ClusterIssuer? 2393
When is an additional ClusterIssuer required? 2393
How to add an additional ClusterIssuer using Let’s Encrypt and DNS-01 verification method? 2394
How to add an additional Issuer and ClusterIssuer using HashiCorp Vault to issue certificates? 2394
How to add ClusterIssuer using its own or intermediate CA to to issue certificates? 2395
How does cert-manager work behind a proxy server? 2396
How to secure cert-manager credentials? 2397
Does the tls-acme annotation work? 2398
How do I check the certificate status? 2399
How do I get a list of certificates? 2400
The “CAA record does not match issuer” error 2400
What to do if the HTTP-01 self-check never completes? 2400
The chrony module 2401
How it works 2401
Boot-time synchronization on master nodes 2401
Limitations 2402
Third-party components 2402
Configuration 2402
Parameters 2404
An example of the configuration 2405
FAQ 2405
How do I disable chrony and use NTP daemon on nodes? 2405
The cilium-hubble module 2406
Requirements 2406
Third-party components 2406
Configuration 2406

Requirements 2408
Conversions 2408
Parameters 2409
Authentication 2411
Cloud provider — AWS 2411
Third-party components 2412
Configuration 2412
Parameters 2414
Provider configuration 2415
AWSClusterConfiguration 2415
Preparing environment 2423
JSON Policy 2424
Configuring IAM via the web interface 2426
Configuring IAM via the CLI 2427
Configuring IAM via Terraform 2428
Custom Resources 2428
AWSInstanceClass 2429
FAQ 2432
How do I create a peering connection between VPCs? 2432
How do I create a cluster in a new VPC with access over an existing bastion host? 2433
How do I create a cluster in a new VPC and set up bastion host to access the nodes? 2433
Configuring a bastion host 2433
Adding CloudStatic nodes to a cluster 2433
How to increase the size of a volume? 2434
How do I configure access to Amazon ECR repository on cluster nodes? 2434
How to create preemptible nodes in AWS? 2434
Handling AWS notifications 2435
Limitations of preemptible nodes in AWS 2435
How do I replace the OS image on the nodes? 2435
What to do if switching to nodes in lower-priority groups takes a long time? 2435
Examples 2436
An example of the AWSInstanceClass custom resource 2436
LoadBalancer 2436
Service object Annotations 2436
Configuring the load balancer if Ingress nodes are not available in all zones 2436
Layouts 2436
WithoutNAT 2437
WithNAT 2438
Cloud provider — Azure 2440
Third-party components 2441
Configuration 2441
Parameters 2443
Provider configuration 2445
AzureClusterConfiguration 2445
Preparing environment 2452
Custom Resources 2453
AzureInstanceClass 2453
Examples 2457
An example of the AzureInstanceClass custom resource 2457
Replacing the OS image 2457
Layouts 2457
Standard 2458
Cloud provider — Basis Dynamix 2458
Third-party components 2459
Configuration 2459
Parameters 2461
Provider configuration 2462
DynamixClusterConfiguration 2462
Preparing environment 2468
Platform version 2468
Account permissions 2468
Storage policies in the cluster resource group 2469

Prepare an operating system image 2469
Replacing the OS image 2469
Custom Resources 2470
DynamixInstanceClass 2470
FAQ 2471
How to configure a LoadBalancer? 2471
How is the disk placement chosen? 2471
Which nodes are recreated when the storage policy changes? 2472
Which StorageClasses does the module create? 2472
Cloud provider - Basis Dynamix: Layouts 2472
Standard 2472
StandardWithInternalNetwork 2473
Cloud provider — GCP 2474
Third-party components 2474
Configuration 2475
Parameters 2477
Provider configuration 2477
GCPClusterConfiguration 2477
Preparing environment 2484
Setup using Google Cloud Console 2484
Setup using gcloud CLI 2484
Custom Resources 2485
GCPInstanceClass 2485
FAQ 2491
How do I create a cluster? 2491
Adding CloudStatic nodes to a cluster 2491
How to create preemptible nodes in GCP? 2491
Limitations of preemptible nodes in GCP 2491
How do I replace the OS image on the nodes? 2492
Examples 2492
An example of the GCPInstanceClass custom resource 2492
Enabling nested virtualization 2492
Adding additional disks 2492
Layouts 2493
Standard 2493
WithoutNAT 2494
Cloud provider — Huawei Cloud 2496
Third-party components 2497
Configuration 2497
List of required Huawei Cloud services 2497
Provider configuration 2497
HuaweiCloudClusterConfiguration 2498
Preparing environment 2503
Configuring IAM via the web interface 2504
JSON policies 2504
Replacing the OS image 2506
Custom Resources 2506
HuaweiCloudInstanceClass 2506
Cloud provider - HuaweiCloud: Layouts 2509
Standard 2509
VpcPeering 2510
Cloud provider — OpenStack 2511
Third-party components 2511
Configuration 2512
List of required OpenStack services 2513
Parameters 2514
Provider configuration 2517
OpenStackClusterConfiguration 2517
Preparing environment 2527
Custom Resources 2527
OpenStackInstanceClass 2528
FAQ 2532

How do I set up LoadBalancer? 2532
IngressNginxController with a pre-created load balancer 2533
IngressNginxController with a pre-allocated floating IP 2533
How do I create NodeGroups in availability zones? 2534
How do I create a hybrid cluster? 2534
Attaching storage devices to instances in a hybrid cluster 2535
How do I create an image in OpenStack? 2535
How do I replace the OS image on the nodes? 2536
How does Cinder CSI handle re-authentication in OpenStack? 2536
How to check whether the provider supports SecurityGroups? 2537
How to set up online disk resize 2537
How to use rootDiskSize and when it is preferred? 2537
Disks in OpenStack 2537
The rootDiskSize parameter 2537
Network disk is recommended for master nodes and bastion host 2537
Local disk is recommended for ephemeral nodes 2537
How do I check the disk volume in a flavor? 2538
How to override a default volume type of cloud provider? 2538
OFFLINE disk resize 2538
How to create preemptible nodes in Selectel? 2538
What to do if switching to nodes in lower-priority groups takes a long time? 2539
Examples 2539
Example 1 2539
Example 2 2539
Cloud provider - OpenStack: Layouts 2540
Standard 2540
StandardWithNoRouter 2542
Simple 2544
SimpleWithInternalNetwork 2546
Cloud provider — VMware Cloud Director 2549
Third-party components 2549
Configuration 2549
List of required VMware Cloud Director services 2549
Parameters 2551
Provider configuration 2553
VCDClusterConfiguration 2553
Preparing environment 2561
List of required VCD resources 2561
User permissions 2562
Adding a network 2565
Configuring DHCP 2568
Adding a vApp 2570
Adding a network to the vApp 2571
Incoming traffic 2572
Configuring DNAT/SNAT rules on the edge gateway 2572
Let’s Encrypt certificate issuance 2573
Configuring a firewall 2574
Virtual machine template 2576
Making a template from an OVA file 2577
Setting up the template in VCD 2581
Replacing the OS image 2586
Using the storage 2587
Using the LoadBalancer 2587
Custom Resources 2587
VCDInstanceClass 2587
Examples 2589
Example configuration of the VCDInstanceClass custom resource 2589
Cloud provider - VMware Cloud Director: Layouts 2589
Standard 2589
WithNAT 2590
Cloud provider — VMware vSphere 2593
Nodes managed by CAPI 2594

vSphere requirement 2594
vCenter tags 2594
Virtual machine placement 2594
Third-party components 2594
Configuration 2594
Conversions 2596
Storage 2596
CSI 2596
Important information concerning the increase of the PVC size 2596
Parameters 2596
Provider configuration 2600
VsphereClusterConfiguration 2600
Preparing environment 2612
Environment requirements 2612
List of required vSphere resources 2612
List of required privileges 2612
vSphere configuration 2615
Configuration in vSphere Client 2615
Creating tags and tag categories in vSphere Client 2615
Configuring Datastore in vSphere Client 2618
Creating and assigning a role in vSphere Client 2618
Configuration via govc 2620
Installing govc 2620
Creating tags and tag categories with govc 2620
Datastore configuration with govc 2620
Creating and assigning a role with govc 2621
VM image requirements 2621
Preparing the virtual machine image 2622
Replacing the OS image 2623
Infrastructure 2623
Networking 2623
Inbound traffic 2623
Using the datastore 2623
Custom Resources 2623
VsphereInstanceClass 2623
Examples 2627
An example of the configuration 2627
An example of the VsphereInstanceClass custom resource 2628
Cloud provider - VMware vSphere: Layouts 2628
Standard 2628
Cloud provider — Yandex Cloud 2630
Third-party components 2631
Configuration 2631
Conversions 2633
Module configuration 2633
Storage 2634
LoadBalancer 2635
Parameters 2635
Provider configuration 2639
YandexClusterConfiguration 2639
Preparing environment 2648
Creating a service account 2648
Credentials Secret 2649
Quotas 2649
Yandex Cloud integration 2649
Configuring security groups 2650
Yandex Lockbox integration 2651
Deployment instructions 2651
Checking that everything works as expected 2652
Yandex Managed Service for Prometheus integration 2653
Writing metrics 2653
Reading metrics with Grafana 2654

- Custom Resources 2654
  - YandexInstanceClass 2654
- FAQ 2659
  - Setting up the INTERNAL LoadBalancer 2659
  - Using a separate target group for a NodeGroup 2659
  - Reserving a public IP address 2660
  - dhcpOptions-related problems and ways to address them 2660
    - Editing parameters 2660
    - Aspects of the use 2660
  - Changing networkType does not recreate CloudEphemeral nodes 2660
  - Setting a custom StorageClass as default 2660
  - Resizing a PVC 2660
  - Adding CloudStatic nodes to a cluster 2663
  - Creating a cluster in a new VPC and setting up a bastion host to access the nodes 2663
  - Long switching to nodes in lower-priority groups 2663
  - Replacing the OS image on the nodes 2663
- Examples 2664
  - An example of the module configuration 2664
  - Using the ru-central1-f zone 2664
    - Using existing subnets 2665
    - Changing zones in a running cluster 2665
  - An example of the YandexInstanceClass custom resource 2665
  - LoadBalancer 2665
    - Service annotations 2665
    - Target group health checks 2666
- Layouts 2666
  - Standard 2666
  - WithoutNAT 2667
  - WithNATInstance 2669
- Cloud provider — zVirt 2671
  - Third-party components 2671
  - Configuration 2671
    - Conversions 2673
    - Parameters 2673
  - Provider configuration 2676
    - ZvirtClusterConfiguration 2676
  - Preparing environment 2682
    - Environment requirements 2682
    - User permissions 2682
    - Prepare an operating system image 2682
    - Prepare a virtual machine template 2683
    - Replacing the OS image 2685
    - Credentials Secret 2685
  - Custom Resources 2686
    - ZvirtInstanceClass 2686
  - Usage 2688
    - Provider configuration 2688
  - FAQ 2689
    - How to get the vNIC profile ID? 2689
    - How to disable individual provider subsystems? 2689
  - Layouts 2689
    - Standard 2690
- The cni-cilium module 2691
  - Limitations 2692
  - Handling external traffic in different bpfLB modes (replacing kube-proxy from Cilium) 2692
  - Operational specifics of CNI Cilium on AWS 2693
  - Using CiliumClusterwideNetworkPolicy 2694
  - Changing Cilium Operation Mode 2694
  - Disabling the kube-proxy Module 2694
  - Using selective load balancing algorithm for services 2694
  - Using Egress Gateway 2694
    - Basic mode 2694

Virtual IP mode 2694
Exporting data from Hubble 2694
Third-party components 2695
Configuration 2695
Parameters 2697
Custom Resources 2703
EgressGatewayPolicy 2703
EgressGateway 2704
HubbleMonitoringConfig 2706
Usage 2717
Cluster peering 2717
Pods of other clusters 2717
Global services 2717
Advanced usage 2718
Cluster peering 2718
Requirements 2719
Enabling 2719
Several neighbours 2719
ClusterNetworkPeer status 2719
Changing settings 2720
Disconnecting a neighbour 2720
Disabling NetworkPeering 2720
Examples 2721
Egress Gateway 2721
Operation principle 2721
Node maintenance 2721
Comparison with CiliumEgressGatewayPolicy 2721
Egress Gateway configuration examples 2721
EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode) 2721
EgressGateway in VirtualIPAddress mode (Virtual IP mode) 2722
EgressGatewayPolicy 2722
HubbleMonitoringConfig 2722
HubbleMonitoringConfig configuration examples 2722
Enabling extended metrics and flow logs export (with filters and field mask) 2722
Collecting Hubble flow logs with the log-shipper module 2723
Enabling BPF trace events on demand 2723
Enable trace events on a single node 2723
Enable trace events cluster-wide 2724
Applying the change 2724
The cni-flannel module 2724
Third-party components 2724
Configuration 2725
Parameters 2726
The cni-simple-bridge module 2727
Managing control plane 2727
Managing certificates 2728
Scaling 2728
Scaling master nodes 2728
Dynamic terminated pod garbage collection threshold 2728
Configuring Control Plane Resource Requests 2728
Version control 2729
Exposing the Kubernetes API 2730
Via Ingress 2730
Via Gateway API 2730
Via a Service of type LoadBalancer 2730
Exposing the Kubernetes API over Ingress 2730
Auditing 2730
Control plane monitoring 2731
Admission plugins enabled by default 2731
Standard admission plugins enabled by Kubernetes 2731
Additional admission plugins enabled by the module 2732
Feature Gates 2733

- Kubernetes 1.33 2733
- Kubernetes 1.34 2734
- Kubernetes 1.35 2734
- Kubernetes 1.36 2734
- Kubernetes 1.37 2734

Protecting sensitive fields in custom resources 2735
Third-party components 2735
Configuration 2736
- Conversions 2737
- Parameters 2737

Custom Resources 2746
- ControlPlaneNode 2746
- ControlPlaneOperation 2751
- KubeSchedulerWebhookConfiguration 2753

FAQ 2755
- How do I add a master node to a static or hybrid cluster? 2755
- How do I add master nodes to a cloud cluster? 2756
- How do I reduce the number of master nodes in a cloud cluster? 2757
- How do I dismiss the master role while keeping the node? 2758
- How do I switch to a different OS image in a multi-master cluster? 2759
  - In a cloud cluster 2759
  - In a static cluster 2760
- How do I switch to a different OS image in a single-master cluster? 2761
  - In a cloud cluster 2761
  - In a static cluster 2761
- How to configure HA mode with two master nodes and an arbiter node? 2761
  - Configuring in a cloud cluster 2762
  - Configuring in a static cluster 2763
- How do I view the list of etcd members? 2763
  - Option 1 2763
  - Option 2 2763
- What if something went wrong? 2764
- What if the etcd cluster fails? 2764
  - What if etcd restarts with an error? 2764
  - What to do if the database volume of etcd reaches the limit set in quota-backend-bytes? 2764
- How to compact etcd storage 2765
  - How to compact etcd storage on a node in a single-master cluster 2765
  - How to compact etcd storage in a cluster with multiple master nodes 2766
- Cluster admin access model 2767
  - Kubeconfig files on master nodes 2767
  - RBAC-based admin access 2767
  - Recommended admin access 2767
  - How to generate a kubeconfig to access the Kubernetes API? 2767
  - Root kubeconfig symlink 2768
  - Security hardening 2768
  - Break-glass access 2768
- How do I configure additional audit policies? 2768
  - How to omit Deckhouse built-in policy rules? 2769
  - How to stream audit log to stdout instead of files? 2769
  - How to deal with the audit log? 2769
- How do I speed up the restart of Pods if the connection to the node has been lost? 2769
- etcd backup and restore 2770
  - What is done automatically 2770
  - How to manually backup etcd 2770
    - Using Deckhouse CLI (Deckhouse Platform v1.65+) 2770
    - Using bash (Deckhouse Platform v1.64 and older) 2770
  - How do perform a full recovery of the cluster state from an etcd backup? 2771
    - Restoring a single-master cluster 2771
    - Restoring a multi-master cluster 2771
  - How do I restore a Kubernetes object from an etcd backup? 2772
    - Example of steps to restore objects from an etcd backup 2772
- How the node to run the Pod on is selected 2774

The structure of the Kubernetes scheduler 2774
Working logic 2774
Scheduler profiles 2774
Pod scheduling stages 2774
Documentation 2775
How to change or extend the scheduler logic 2775
How to check which node a Pod would be scheduled to 2775
How to monitor the scheduler 2776
How does kubelet certificate rotation work? 2776
Certificates lifetime 2776
Specifics of working with kubelet server certificates in Deckhouse Platform 2776
How to manually update control plane component certificates? 2777
How do I protect sensitive fields in custom resources? 2777
Protection mechanisms 2777
Built-in audit rules 2778
What is Audit Policy 2778
Policy structure 2778
Built-in audit rules 2778
Full policy example 2785
Examples 2791
The connection of an external scheduler plugin 2791
Protecting resources with sensitive fields 2791
The csi-vsphere module 2793
Main Features 2794
System requirements 2794
Architecture 2794
Compatibility modes 2794
Storage discovery and provisioning 2794
Default StorageClass and exclusions 2794
Volume lifecycle 2795
Limitations 2795
Configuration 2795
Parameters 2796
Preparing environment 2799
Required resources 2799
Installing govc 2799
Create tags and tag categories 2799
Datastore configuration 2799
Create and assign the role 2799
The deckhouse module 2800
Deckhouse releases update 2800
Get Deckhouse releases status 2800
Update process 2800
Manual release deployment 2800
Pin a release 2800
Priority Classes 2800
Monitoring 2801
Metrics collection 2801
Integration with the observability module 2801
Third-party components 2801
Configuration 2802
Conversions 2803
Parameters 2803
Custom Resources 2811
ConversionWebhook 2811
ValidationWebhook 2813
Usage 2816
Setting up the update mode 2816
Update windows configuration 2816
Manual update confirmation 2816
Deckhouse update notifications 2817
Reserving the hostnames of the platform web interfaces 2818

Reserved hostnames 2818
Excluding a name from reservation 2818
Excluding the shared Gateway 2819
Reservation with existing applications in the cluster 2819
Checking reservation status 2819
Collect debug info 2820
FAQ 2820
How to run kube-bench in my cluster? 2820
How to collect debug info? 2820
How to debug pod problems with ephemeral containers? 2821
Who can run exec into Deckhouse Platform pods and scale Deckhouse Platform objects? 2821
How to debug node problems with ephemeral containers? 2821
The deckhouse-tools module 2822
Configuration 2822
An example of the configuration 2822
Parameters 2823
The descheduler module 2825
Features of the module 2825
Metrics provider 2825
Evictions in background 2825
Strategies 2825
HighNodeUtilization 2825
LowNodeUtilization 2826
CPU utilization averaged over 15 minutes 2826
Eviction target check 2826
RemoveDuplicates 2826
RemovePodsHavingTooManyRestarts 2826
RemovePodsViolatingInterPodAntiAffinity 2826
RemovePodsViolatingNodeAffinity 2827
RemovePodsViolatingTopologySpreadConstraint 2827
Third-party components 2827
Configuration 2827
Conversions 2828
Parameters 2828
Custom Resources 2828
Descheduler 2829
Examples 2840
Configuring the pod redistribution interval 2840
Example LowNodeUtilization strategy 2840
Example LowNodeUtilization strategy with thresholds relative to the average 2840
Example LowNodeUtilization strategy on the 15-minute CPU average 2840
Example HighNodeUtilization strategy 2840
Protecting pods by StorageClass 2841
Module documentation 2841
Main features 2841
How the module works 2841
Limitations 2841
The documentation module: configuration 2842
Parameters 2843
The documentation module: usage 2845
An example of the module configuration 2845
Authentication 2846
S3 storage of your own 2846
High availability mode 2846
The infra-provider-baremetal module 2847
Key features 2847
How it works 2847
Requirements 2847
Limitations 2847
Third-party components 2847
Configuration 2848
Requirements 2849

- Parameters 2849
- Custom resources 2852
  - BareMetalImage 2852
  - BareMetalInstance 2853
  - BareMetalInstanceClass 2856
  - BareMetalIPXEFirmware 2857
  - BareMetalRamdiskImage 2858
- Usage 2858
  - Choosing the Ironic mode 2858
    - Ironic in the cluster 2858
    - External Ironic 2859
  - Initializing Kubernetes nodes 2859
  - Registering physical servers 2859
    - Connecting to the BMC 2859
    - Server parameters 2859
    - Checking server state 2859
  - Creating nodes 2859
  - Enabling TLS for server boot 2860
- Examples 2860
  - Creating a node with Ironic in the cluster 2860
  - Connecting to an external Ironic 2861
  - Enabling TLS for server boot 2862

The istio module 2863

- Compatibility table for supported versions 2863
- Problems Istio helps to solve 2863
- Mutual TLS 2863
- Authorization 2863
- Request routing 2863
- Managing request balancing between service Endpoints 2863
- Observability 2864
  - Tracing 2864
  - Grafana 2864
  - Kiali 2864
  - Mesh metrics and access logs (Telemetry API) 2864
- Architecture of the cluster with Istio enabled 2864
- Application service architecture with Istio enabled 2865
  - Details 2865
  - Native sidecars 2865
  - User request lifecycle 2865
    - Application with Istio turned off 2865
    - Application with Istio turned on 2865
- Activating Istio to work with the application 2865
- Federation and multicluster 2866
  - Federation 2866
    - Requirements for clusters 2866
    - General principles of federation 2866
    - Enabling the federation 2866
    - Managing the federation 2866
  - Multicluster 2867
    - Requirements for clusters 2867
    - General principles 2867
    - Enabling the multicluster 2867
    - Managing the multicluster 2867
- Ambient mesh 2867
  - Components 2867
  - Prerequisites 2867
  - Ambient mesh limitations 2868
  - Enrolling workloads 2868
- Authentication 2868
- Estimating overhead 2868
  - control-plane 2868
  - data-plane 2868

- Third-party components 2868
- Configuration 2869
  - Requirements 2870
  - Conversions 2870
  - Parameters 2870
- Custom Resources 2888
- IngressIstioController 2888
- IstioFederation 2892
- IstioMulticluster 2895
- WaypointInstance 2898
- Custom Resources (by istio.io) 2902
- Routing 2902
  - DestinationRule 2902
  - VirtualService 2903
  - ServiceEntry 2903
- Authentication 2903
  - PeerAuthentication 2903
  - RequestAuthentication 2903
- Authorization 2903
  - AuthorizationPolicy 2903
  - Sidecar 2904
- Examples 2904
- Circuit Breaker 2904
- gRPC balancing 2905
- Locality Failover 2905
- Retry 2906
- Canary 2906
  - Cookie-based routing 2907
  - Probability-based routing 2907
- Ingress to publish applications 2908
  - Istio Ingress Gateway 2908
    - Preserving client attributes behind external proxies 2909
    - Managing gateway resource requests 2910
  - Ingress NGINX 2911
- Authorization configuration examples 2912
  - Decision-making algorithm 2912
  - Deny all actions for the foo namespace 2913
  - Deny requests from the foo NS only 2914
  - Allow requests for the foo NS only 2914
  - Allow requests from anywhere in the cluster 2914
  - Allow any requests for foo or bar clusters 2914
  - Allow any requests only from entities in the baz namespace of the foo or bar clusters 2915
  - Allow from any cluster (via mTLS) 2915
  - Allow all requests from anywhere (including no mTLS - plain text traffic) 2915
- Setting up federation for two clusters using the IstioFederation CR 2915
- Setting up multicluster for two clusters using the IstioMulticluster CR 2916
- Ambient mesh 2916
  - Enabling ambient mesh 2916
  - Enrolling workloads into the ambient (L4) mesh 2917
  - Adding a waypoint for L7 features 2917
  - Disabling ambient mesh 2918
- Control the data-plane behavior 2918
  - Jobs with an Istio sidecar 2918
  - Prevent istio-proxy from terminating before the main application’s connections are closed 2918
- Telemetry API for mesh metrics and access logs 2918
  - Enabling Telemetry API mode 2919
  - Verifying metrics and logs 2919
  - Prometheus scraping and Grafana 2919
  - Extra Telemetry policies (optional) 2920
  - Tracing with Telemetry API 2920
    - Example — Telemetry API bundle + bundled Zipkin/Jaeger collector 2920
    - Kiali 2920

Example — mesh-wide OTLP via ModuleConfig 2921
Example — tracing only for selected workloads 2921
Example — disable span export for ingress-only namespaces 2922
Rolling back to the legacy telemetry stack 2922
Debugging Istio with istioctl from the debug container 2922
CNIPlugin application traffic redirection mode restrictions 2924
Upgrading Istio 2924
Upgrading Istio control-plane 2924
Auto upgrading istio data-plane 2925
Customizing istio-proxy sidecar resource management 2926
Supported annotations 2926
Configuration Examples 2926
Providing a custom CA via a Secret reference 2927
Example: self-signed CA issued by cert-manager 2927
Example: intermediate CA issued by a Vault issuer 2928
Reverting to a self-signed CA 2929
The keepalived module 2930
Third-party components 2930
Configuration 2930
Custom Resources 2932
KeepalivedInstance 2932
keepalived: FAQ 2935
How to manually switch keepalived? 2935
Examples 2935
Three public IP addresses 2935
The kube-dns module 2937
Third-party components 2937
Configuration 2937
Parameters 2939
FAQ 2941
How to change cluster domain with minimal downtime? 2941
How to scale-up kube-dns replicas? 2943
Examples 2943
Configuration example 2943
The kube-proxy module 2944
The metallb module 2944
Layer 2 mode 2945
Advantages of the module over the classic MetalLB 2945
BGP Mode 2945
How metallb Works in BGP Mode 2945
Advantages of Using BGP 2946
Disadvantages of Using BGP 2946
Third-party components 2946
Configuration 2946
Conversions 2948
Parameters 2948
Custom Resources 2952
MetalLoadBalancerBGPPeer 2952
MetalLoadBalancerClass 2954
MetalLoadBalancerConfiguration 2955
MetalLoadBalancerPool 2956
Examples 2956
Example of metallb usage in L2 LoadBalancer mode 2957
Running speakers on nodes with custom taints 2958
Example of metallb usage in BGP LoadBalancer mode 2958
Example of using L2 LoadBalancer and BGP LoadBalancer modes simultaneously 2959
Limitations when running the modes simultaneously 2961
Example of migration from L2 LoadBalancer mode to BGP LoadBalancer mode 2962
Before you begin 2962
Step 1. Binding L2 services to MetalLoadBalancerClass 2962
Step 2. Disabling the default MetalLoadBalancerClass 2963
Step 3. Enabling BGP LoadBalancer mode 2963

Step 4. Moving a single-address service to BGP LoadBalancer mode 2964
Additional configuration examples for Service 2965
Creating a Service and assigning it specific IP addresses from the pool 2965
Creating a Service with a specific IP address 2965
Creating Services with shared IP addresses 2966
Assigning an IP address pool to a Service in BGP LoadBalancer mode 2967
Module multitenancy-manager 2968
Main features 2968
Project composition 2968
How the module works 2968
Creating a project 2968
Multiple namespaces in a project 2969
Access to a project 2969
Automatic project creation for namespaces 2969
Isolating a project 2969
Managing access to cluster-wide resources 2970
How access management works 2970
Access checks and default value assignment 2970
Determining cluster-wide resource availability 2971
Assigning default values 2971
Checking existing objects 2971
System requests 2971
Monitoring access policy violations 2971
Resources registered by Deckhouse Platform 2971
Access management resources 2974
The multitenancy-manager module: configuration 2974
Parameters 2976
The multitenancy-manager module: custom resources 2977
ClusterProjectRoleBinding 2977
AvailableClusterResource 2978
ClusterResourceGrantPolicy 2979
GrantableClusterResourceDefinition 2987
GrantableClusterResourceReference 2992
ProjectNamespace 2996
ProjectQuotaStatus 2997
ProjectRoleBinding 2999
Project 3000
ProjectTemplate 3008
The multitenancy-manager module: usage examples 3015
Default project templates 3015
Creating a project 3016
Project API versions 3017
Creating a project without specifying a template 3017
Project naming rules 3017
Project status and diagnostics 3018
Service objects of a project 3018
Project creator and author of the last change 3019
Virtual projects 3019
Additional project namespaces 3019
What applies to the additional namespaces 3020
Labels of the project namespaces 3021
Creating a project automatically for a namespace 3022
Standard project fields 3024
Project quota 3025
Keys of a project quota 3026
Virtual machines in a project quota 3027
Load balancers in a project quota 3027
Clusters where Deckhouse Platform does not manage the control plane 3028
Dividing a project quota between namespaces 3028
Who manages the quotas 3028
When the project quota refuses 3028
Granting access within a project 3030

Roles available in a RoleBinding inside a project 3031
Structured templates 3032
Template parametrization 3033
Template checks 3034
Describing other objects in manifests 3035
Additional namespaces in manifests 3037
Limitations of manifests 3037
Moving a template from a Helm text 3038
Switching a project from a Helm text 3042
Moving an object of the release to manifests 3043
Creating your own project template 3043
Labels for the objects of a Helm text 3044
Skipping the heritage label 3044
Excluding objects from management 3044
Implementing validation of object changes with a custom label 3045
How validation works in multitenancy-manager 3045
Who bypasses admission 3045
Creating your own validation 3046
Managing access to cluster-wide resources 3048
How a policy selects namespaces 3048
For cluster administrators 3049
Restricting StorageClass for a project 3049
Restricting ClusterIssuer for a project 3050
Restricting ClusterRoles for a project 3051
Restricting LoadBalancerClass for a Service 3051
Granting access to all resources of a specific type 3052
Granting a reserved resource 3052
Denying individual resources 3054
Managing access using label selectors 3054
Granting cluster-wide resources through a project template 3055
Granting cluster-wide resources to one project 3055
For project users 3056
Viewing available cluster-wide resources 3056
Rejection due to an unavailable cluster-wide resource 3057
Assigning default values automatically 3057
Projects without a default value 3057
For module developers 3058
Configuring validation of a cluster-wide resource reference 3058
Registering a new cluster-wide resource 3059
Showing resource fields to projects 3060
Using x-deckhouse-grantable-resource in Deckhouse Platform application settings 3061
Checking resource registration status 3061
The multitenancy-manager module: FAQ 3062
Managing access to cluster-wide resources 3062
What should I do if a PersistentVolumeClaim is rejected with an “is not available to project” error? 3062
How can I view access policies applied to a project? 3062
What happens to existing objects when access is restricted? 3062
What does the ClusterResourceGrantPolicyViolation alert mean? 3063
How can I allow all StorageClasses except specific ones? 3063
What happens if there is no ClusterResourceGrantPolicy? 3063
I created a ClusterResourceGrantPolicy, but nothing changed — why? 3063
How does cluster-wide resource access management interact with RBAC? 3064
What happens if the multitenancy-manager module is disabled? 3064
The namespace-configurator module 3064
How does it work? 3064
What do I need to configure? 3064
Configuration 3064
Parameters 3066
Examples 3067
Example 3067
The network-gateway module 3068

Configuration 3068
Parameters 3070
The network-policy-engine module 3072
Third-party components 3072
Configuration 3073
Examples 3074
Examples of network policies 3075
Deny external traffic to Pods in the namespace, but allow traffic to external resources from within the namespace 3075
The node-local-dns module 3075
Module Description 3075
Purpose 3075
Grafana dashboard 3076
How does it work? 3076
Aspects of CoreDNS configuration 3076
Configuration 3076
Requirements 3078
Parameters 3078
Examples 3079
An example of configuring a custom DNS for a Pod 3079
Configuration example 3080
Node-manager module 3080
Main features 3080
Node types 3081
Node grouping and group management 3081
Delaying node reboot or shutdown while critical pods are running 3081
How it works and key features 3082
Delaying mechanism features in Deckhouse Platform 3082
Automatic deploying, configuring and updating Kubernetes nodes 3082
Deploying Kubernetes nodes 3082
Keeping nodes up-to-date 3082
Working with nodes on supported cloud platforms 3083
Scaling nodes in the cloud 3083
Working with static nodes 3083
Cluster API Provider Static 3084
Custom node settings 3085
Features of writing scripts 3088
Chaos Monkey 3088
Monitoring 3088
What information does Prometheus collect and in what form? 3088
Third-party components 3088
Managing nodes: configuration 3089
Conversions 3090
Parameters 3091
Managing nodes: custom resources 3093
FencingFailedNodeState 3093
FencingSLAProfile 3097
Instance 3101
InstanceTypesCatalog 3105
NodeGroup 3106
NodeConfig 3144
NodeExtensionRequest 3166
NodeGroupConfiguration 3171
NodeOperation 3172
NodeStaticPodRequest 3177
NodeUser 3181
SSHCredentials 3184
StaticInstance 3187
Managing nodes: FAQ 3194
How do I add master nodes to a cloud cluster? 3194
How do I reduce the number of master nodes in a cloud cluster? 3194
Static nodes 3194

How do I add a static node to a cluster (Cluster API Provider Static)? 3194
How do I add a batch of static nodes to a cluster manually? 3195
How do I clean up a static node manually? 3196
For all nodes except the control-plane 3197
For control-plane nodes 3197
Can I delete a StaticInstance? 3197
How do I change the IP address of a StaticInstance? 3198
How do I migrate a manually configured static node under CAPS control? 3198
Why are the SSH key and the sudo password of SSHCredentials shown as <omitted>? 3198
How do I change the NodeGroup of a static node? 3198
How to clean up a node for adding to the cluster? 3199
How do I know if something went wrong? 3199
How do I know what is running on a node while it is being created? 3199
How do I update kernel on nodes? 3200
Debian-based distros 3200
CentOS-based distros 3200
NodeGroup parameters and their result 3201
How do I redeploy ephemeral machines in the cloud with a new configuration? 3202
How do I allocate nodes to specific loads? 3202
How do I give pods dedicated CPUs and NUMA alignment? 3202
Rules of the Restricted and SingleNumaNode policies 3204
Changing the settings 3204
Limitations 3204
How to allocate nodes to system components? 3205
Frontend 3205
System components 3205
How do I speed up node provisioning on the cloud when scaling applications horizontally? 3205
How do I disable machine-controller-manager/CAPI in the case of potentially cluster-damaging changes? 3205
How do I restore the master node if kubelet cannot load the control plane components? 3206
containerd 3206
How to change CRI for NodeGroup? 3206
Why might the CRI change not apply? 3207
How to change CRI for the whole cluster? 3207
How to add node configuration step? 3208
How to automatically put custom labels on the node? 3208
How to deploy custom containerd configuration? 3209
How to add configuration for an additional registry? 3210
Old Method 3211
How to add additional registry auth (deprecated method)? 3211
How to configure a certificate for an additional registry (deprecated method)? 3212
How to add TLS skip verify (deprecated method)? 3213
How to set up a mirror for public image registries (deprecated method)? 3214
New Method 3215
How to add additional registry auth (actual method)? 3216
How to configure a certificate for an additional registry (actual method)? 3217
How to add TLS skip verify (actual method)? 3217
How to set up a mirror for public image registries (actual method)? 3218
How to use NodeGroup’s priority feature 3219
How to interpret Node Group states? 3220
How do I make werf ignore the Ready conditions in a node group? 3222
What is an Instance resource? 3222
When is a node reboot required? 3222
How to enable a delay before a node shutdown or restart while critical pods are running on it? 3222
Searching pods that prevent shutdown and performing the node troubleshooting 3223
How do I set up fencing for a node group? 3223
How do I check the fencing state of the nodes? 3224
What happens when the fencing agent restarts? 3225
What are the limits of softdog? 3225
Deckhouse Engine nodes 3225
How does fencing work on master nodes? 3225

- Fencing in a cluster with one master node 3225
- Which network ports does fencing need? 3226
- What does the D8FencingSystemNetworkNotFound alert mean? 3226
- How does fencing protect the network key? 3226
- How do I rotate the fencing network key? 3226
- How the fencing mechanism handles different node types? 3227
  - When the controller sets the taint 3228
  - Phases of a fencing incident 3229
  - Fencing alerts 3229
  - Known limitations of the evacuation 3230
- What to do with the out-of-service taint before you turn fencing off in all node groups? 3231
- How do I work with GPU nodes? 3231

Managing nodes: examples 3232
- Examples of the NodeGroup configuration 3232
  - Cloud nodes 3232
  - Static nodes 3232
  - System nodes 3232
  - Nodes with GPU 3233
- Adding a static node to a cluster 3233
  - Manually 3233
  - Using the Cluster API Provider Static 3234
  - Using Cluster API Provider Static for multiple node groups 3236
  - Cluster API Provider Static: Moving Instances Between Node Groups 3237
    - Initial Configuration 3237
    - Steps to Move an Instance Between Node Groups 3238
    - 1. Create a New NodeGroup for the Target Node Group 3238
    - 2. Update the Label on the StaticInstance 3238
    - 3. Decrease the Number of Static Instances in the Original NodeGroup 3238
- An example of the NodeUser configuration 3239
- An example of the NodeGroupConfiguration configuration 3239
  - Installing the cert-manager plugin for kubectl on master nodes 3239
  - Tuning sysctl parameters 3239
  - Adding a root certificate to the host 3239
  - Adding the ability to download images from insecure container registry to containerd 3241

The okmeter module 3241
- Configuration 3241
  - Parameters 3243
- Examples 3244
  - Example 3244

The openvpn module 3245
- Exposing the VPN service 3245
- Available cluster resources after connecting to the VPN 3245
- User’s traffic audit 3245
- Third-party components 3245
- Configuration 3246
  - Conversions 3247
  - Parameters 3248
  - Authentication 3252
- FAQ 3252
  - Why doesn’t automatic DNS server configuration work when connecting to macOS and Linux using the OpenVPN client 3252
  - How to revoke, rotate or delete a user certificate 3253
  - How to rotate a server certificate 3253
  - How to rotate a root certificate (CA) 3253
- Examples 3254
  - An example for bare metal clusters 3254
  - An example for AWS & Google Cloud 3254
  - An example for an external load balancer with a public IP address 3254

Registry Module 3256
- Current implementation 3256
  - Modes of operation 3256
    - Installer behavior when no container image registry is configured 3257

Additional registries 3257
Mirrors for remote nodes 3258
Architecture 3258
The node agent 3258
The in-cluster cache 3258
The expected image set 3259
Replication 3259
Leader election 3260
What protects the ports on control-plane nodes 3261
Air-gapped clusters 3262
Installing from a bundle 3262
Requirements for using the module 3263
The v1 implementation 3263
Mode switching restrictions 3264
Direct mode architecture 3264
Proxy mode architecture 3264
Local mode architecture 3265
Module registry: configuration 3265
Parameters 3267
The registry module: Custom Resources 3274
RegistryConfig 3274
RegistryMirror 3279
RegistryNode 3283
RegistryOperation 3290
RegistryStorage 3296
RegistryUpstream 3306
FAQ 3309
How does the migration to the registry module work? 3310
Which implementation is my cluster running? 3310
How do I migrate from Unmanaged mode? 3311
How do I migrate from Direct mode? 3311
How do I migrate from Proxy mode? 3312
How do I migrate an air-gapped cluster from Local mode? 3312
Disk requirements 3312
Procedure 3313
Does the migration restart containerd? 3314
What do the registry alerts mean? 3314
How do I remove leftover cache data from a node? 3316
The cache keeps growing. What reclaims it? 3316
When it runs, and what happens during a run? 3317
Is a collection worth running now? 3318
How to run garbage collection manually? 3318
Turning it off 3319
What happens when the cache’s store is full? 3319
The store’s disk did not mount. What happens? 3320
A pull is failing on a node. Where do I look? 3320
Modules from other ModuleSources: what the cache holds and what to do before air-gap 3321
What does destroying an air-gapped cloud cluster need from the store? 3322
How do I push a new Deckhouse Platform release or a new module version into an air-gapped cluster? 3322
Step 1. Pull the images on a machine with internet access 3322
Step 2. Check the publication endpoint 3323
Step 3. Push the images 3323
Step 4. Check that the release has reached the cluster 3323
Image availability alerts from extended-monitoring for the in-cluster registry 3324
How do I check the state of the in-cluster cache? 3324
Which images are in the cache? 3325
Which images take the most space? 3325
How do I check and repair images in the cache? 3326
How is the cache updated? 3327
One master node in an air-gapped cluster 3327
Several master nodes in an air-gapped cluster 3327

What the UpdateBlocked event means 3327
Who can read from and write to the in-cluster cache? 3328
The v1 implementation 3328
How to Migrate to the registry module? 3328
For containerd v2 3328
For Containerd v1 3329
How to Migrate Back from the Registry Module? 3333
How to check the registry mode switch status? 3335
Module registry: usage examples 3337
Enabling the module 3337
Turning the in-cluster cache on 3338
Going air-gapped 3339
Adding another registry 3340
Pulling from a mirror near the nodes 3341
Pulling from a private registry without declaring it 3341
Turning the module off again 3342
Examples for the v1 implementation 3342
Switching to the Unmanaged Mode 3342
The registry-packages-proxy module 3343
Architecture 3344
HTTP API 3344
Package icons (/v1/packages/) — in-cluster only 3344
Deckhouse CLI downloads (/v1/images/) — public 3345
Internal /package endpoint 3346
Package retrieval flow 3346
High availability 3346
RBAC roles provided by the module 3346
Granting access to Deckhouse CLI downloads 3346
Limitations 3347
The registry-packages-proxy: configuration 3348
Parameters 3351
The secret-copier module 3352
How does it work? 3352
What do I need to configure? 3352
How to synchronize Secret to some selected namespaces instead of all namespaces? 3353
How do I know that a Secret was not copied? 3353
The terraform-manager module 3353
Third-party components 3353
Configuration 3353
Parameters 3355
Module user-authn 3356
Main features 3356
Components 3356
Integration features 3357
Basic authentication for the Kubernetes API requests 3357
Application integration 3357
Generating kubeconfig files 3357
Additional Dex capabilities 3358
High availability mode 3358
Third-party components 3358
The user-authn module: configuration 3358
Conversions 3360
Parameters 3361
The user-authn module: Custom Resources 3376
DexAuthenticator 3376
DexClient 3396
DexProviderCheck 3401
DexProvider 3402
Group 3430
TokenExchangeProvider 3431
UserAccount 3435
UserOperation 3439

- User 3444
- The user-authn module: usage 3449
  - An example of the module configuration 3449
  - Configuring a provider 3449
    - Checking provider connectivity 3449
    - Limiting the identities of a provider 3450
    - Prefix for the groups of a provider 3451
    - GitHub 3452
    - GitLab 3452
    - Atlassian Crowd 3453
    - Bitbucket Cloud 3453
    - OIDC (OpenID Connect) 3453
      - Keycloak 3454
      - Okta 3455
      - Blitz Identity Provider 3455
    - LDAP 3456
      - Nested groups 3456
      - Signing in with one of several attributes 3457
      - Configuring Basic Authentication 3457
      - Kerberos (SPNEGO) SSO for LDAP 3458
    - SAML 3459
  - Configuring the OAuth2 client in Dex for connecting an application 3460
    - Granting an application access to the Kubernetes API 3461
    - Limiting the providers of an application 3461
  - Exchanging tokens of other issuers 3462
  - Browser sessions of Dex 3464
    - How long a session lasts 3464
    - Signing out of the application of a DexClient 3465
  - Local Authentication 3465
    - Creating a user 3465
    - Deleting a user 3467
    - Local user operations 3467
      - Administrative operations 3467
      - Self-service operations 3468
      - Creating UserOperation manually 3469
      - Operations on external users 3471
      - Users of a deleted provider 3471
      - UserOperation lifecycle and side effects 3472
      - Checking operation status 3472
      - Viewing account state 3472
      - Automatic lockout 3473
    - Adding a user to a group 3473
    - Viewing users and groups 3474
    - Password policy 3474
    - Two-factor authentication (2FA) 3476
      - Security keys (WebAuthn) 3477
      - Several methods 3477
      - The second factor of an account 3478
    - Assigning permissions to a user or group 3479
- The user-authn module: FAQ 3479
  - How to secure my application? 3479
    - Setting up CIDR-based restrictions 3481
    - Checking that a DexAuthenticator is served 3482
    - Updating from a version with a dex-authenticator for every DexAuthenticator 3482
  - How to pass the user’s login and groups to an application? 3482
    - JWT token contents and processing considerations 3484
  - Authentication flow with DexAuthenticator 3484
    - Error 500 while Dex is unavailable 3484
    - Handling an expired session in a single-page application 3484
    - A sign-in that took longer than 10 minutes 3485
    - Ending a DexAuthenticator session 3485
  - How to generate a kubeconfig and access Kubernetes API? 3485

Configuring kube-apiserver 3485
The flow of accessing Kubernetes API with generated kubeconfig 3485
How to rotate the secret of the kubernetes OAuth2 client? 3486
How to enable Kerberos (SPNEGO) SSO for LDAP? 3486
How to configure Basic Authentication for accessing Kubernetes API via LDAP? 3486
How is Dex protected against credential brute-forcing? 3487
Which client address does Dex see? 3487
How to find out why a sign-in was refused? 3487
A UserOperation is in the Failed status — what should I do? 3488
How do I unlock a user? 3488
A user was locked automatically — why? 3488
Can I delete a UserIdentity object? 3488
Can I cancel a UserOperation? 3488
The user-authz module 3488
Granular role-based model 3489
Role scopes 3489
Access levels 3490
Using ClusterAuthorizationRule and AuthorizationRule together with RBAC 3490
Namespace roles 3490
Automatic access to cluster-wide catalog resources 3491
Project roles 3491
System and subsystem roles 3492
Access of system and subsystem roles to the namespaces of the modules 3493
Who can assign roles 3493
Global resource dictionaries 3494
Automatic binding 3495
Subsystems of the role-based model 3495
How the roles are built: aggregation and capabilities 3497
Reference of role labels and annotations 3497
Admin level restrictions and superadmin rights 3499
Built-in protection mechanisms of the role model 3500
Display names of roles 3500
Deprecated role names 3500
Basic role-based model 3503
Default access list for each role 3504
Configuration 3510
Parameters 3512
Custom Resources 3513
AuthorizationRule 3513
ClusterAuthorizationRule 3515
Usage 3525
Example of assigning rights to a cluster administrator 3525
Example of assigning rights to a network administrator 3525
Example of assigning administrative rights to a user within a namespace 3527
An example of assigning rights across all namespaces of a project 3528
An example of a ClusterAuthorizationRule 3529
Managing permissions via CLI 3530
Example of granting access to all namespaces 3531
Creating a user 3532
Creating a ServiceAccount for a machine and granting it access 3532
How to create a user using a client certificate 3535
Creating a user using a certificate issued via OpenSSL 3535
Creating a user by issuing a certificate via the Kubernetes API 3536
Granting access to the created user 3538
Configuring kube-apiserver for multi-tenancy mode 3538
How do I check that a user has access? 3539
Customizing rights of high-level roles 3540
FAQ 3541
How do I create a user? 3541
How do I limit user rights to specific namespaces? 3541
What if there are two ClusterAuthorizationRules matching a single user? 3542
Can the basic and the granular role-based models be used at the same time? 3542

- How do I get an equivalent of the ClusterAdmin and SuperAdmin roles in the granular model? 3543
- How do I grant a user access to the resources of one module only? 3544
- What happens to the bindings when the cluster is updated to the release with the controller? 3545
- How do I check that the controller keeps the bindings in sync? 3546
- How do I check that the authorization webhook has the current rules? 3547
- Why does a change to a ClusterAuthorizationRule take up to 30 seconds to take effect? 3549
- Why is a rule reported as needing multi-tenancy? 3549
- How do I extend a role or create a new one? 3550
  - Creating a new role subsystem 3550
  - Extending the custom role 3551
  - Extending the existing subsystem roles 3552
  - Extending subsystem roles and adding a new namespace 3553
  - Extending the existing namespace roles 3553
  - Creating a custom namespace or project role 3554
- How do I migrate custom roles to the new scheme? 3555
  - Fixing a role 3558
    - Roles aggregated into the system roles 3559
    - Selectors by a lineage no built-in role collects 3560
  - Replacing a capability 3560
  - Roles frozen earlier 3561
  - Examples 3562
    - Custom role before and after 3562
    - Custom capability before and after 3564
  - Labels and annotations: before and after 3564
  - Adding a custom capability (in the new scheme) 3566
- How do I rename a built-in role? 3566
- How do I find out who has access to a resource? 3567
- How do I find out what a specific user, group, or ServiceAccount is allowed to do? 3567
- How does a user see the list of namespaces available to them? 3568
- How does an interface find out where a user may perform an action? 3568

The vertical-pod-autoscaler module 3572

- Grafana dashboard 3572
- Vertical Pod Autoscaler Architecture 3572
- Third-party components 3573
- The vertical-pod-autoscaler: configuration 3573
  - Requirements 3575
  - Parameters 3575
- Custom resources 3576
  - VerticalPodAutoscaler 3576
  - VerticalPodAutoscalerCheckpoint 3587
- FAQ 3590
  - How do I view the Vertical Pod Autoscaler recommendations? 3590
  - How does Vertical Pod Autoscaler handle limits? 3590
    - Example No. 1 3590
    - Example No. 2 3591
  - How do I make VPA recommend more or less than it measured? 3591
- Examples 3592
  - The module configuration 3592
  - The basic VerticalPodAutoscaler custom resource example 3593
  - The advanced VerticalPodAutoscaler custom resource example 3593

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse Platform documentation

Download PDF

Welcome to the Deckhouse Platform documentation homepage. Deckhouse Platform is an AI-driven platform for building hybrid distributed infrastructure.

Here you can find information about installing, configuring, and using the platform, as well as descriptions of its features, components, and modules.

If you are new to Deckhouse Platform, we recommend starting with the “Getting Started” section, which provides step-by-step instructions for deploying the platform on any infrastructure.

To quickly find the information you need:

- Use search if you are looking for a specific Deckhouse Platform feature, parameter, or resource.
- Go to the list of modules if you are looking for information about a specific module.
- Use the sidebar to navigate through the documentation sections.

The content of the Deckhouse Platform documentation may vary depending on the version. Select the required version from the drop-down list at the top of the page. The list contains currently supported documentation versions.

If you need help:

- Ask a question in the “Deckhouse | EN-community” Telegram channel.
- If you use a commercial edition of Deckhouse Platform, contact technical support at `support@deckhouse.io`.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

System requirements

This page describes the requirements for cluster nodes and the environment that must be met before installing Deckhouse Platform.

Before the installation, check the following:

- The deployment option is chosen: how many nodes the cluster has, how roles are distributed between them, and whether the cluster tolerates a node failure.
- Nodes have the CPU, memory and disks from the tables for the chosen deployment option and workload type: running containers, running virtualization or combined workloads.
- Nodes have fast disks (400+ IOPS), and the disk layout follows the disk layout and size recommendations.
- Nodes meet the node requirements: CPU architecture, operating system, kernel version and installed software.
- For an installation into an existing Kubernetes cluster, its version is in the list of supported Kubernetes versions, and the node OS is in the list of supported OS.
- The network between nodes meets the network requirements: nodes have permanent IP addresses, the required ports are open, latency between master nodes is within limits, and a domain for the platform web interfaces is prepared.
- If the cluster will run virtual machines, nodes and the infrastructure meet the requirements for running virtual machines.
- There is access to the infrastructure: to the container image registry, to nodes over SSH or to the cloud provider API, and the installer host is ready to run the installation.

Deployment options

The deployment option defines how many nodes the cluster has, how roles are distributed between them, and whether the cluster tolerates a node failure.

A cluster can have nodes of the following types:

- **Master nodes** — nodes that run the control plane components and manage the cluster.
- **Worker nodes** — nodes for the user workload: containerized applications and virtual machines.
- **System nodes** — nodes for Deckhouse Platform modules.
- **Frontend nodes** — nodes that balance incoming traffic. Ingress controllers run on them.
- **Monitoring nodes** — nodes for Prometheus, Grafana and other monitoring components.

For details on configuring nodes of each type, see Things to consider when configuring in the “Going to production” guide.

Examples of deployment options:

- **Single-node cluster** (Single Node / Edge). One node performs all roles: it runs the control plane, Deckhouse Platform modules and the user workload. This option suits getting acquainted with the platform, test environments and edge scenarios.

  > ⚠ A single-node cluster is not fault-tolerant. Do not use it in production environments.

- **Single-master cluster**. One master node manages the cluster, and the user workload runs on one or more worker nodes. Some Deckhouse Platform components also run on worker nodes. This option suits small projects with a low load. The cluster is not fault-tolerant: if the master node fails, the entire cluster is affected.
- **High-availability (HA) cluster with three master nodes**. The control plane is distributed across three master nodes and keeps working if one of them fails: the two remaining nodes keep the quorum. The user workload can run on master nodes or on dedicated worker nodes. This is the recommended option for production environments. To provide high availability with limited resources, you can use an arbiter node instead of one of the master nodes.
- **Cluster with role separation**. The control plane runs on three dedicated master nodes, Deckhouse Platform modules run on system nodes, and Ingress controllers run on frontend nodes. The user workload runs only on worker nodes and is isolated from system components. In loaded clusters with a large volume of metrics, monitoring components are moved to dedicated monitoring nodes; without them, they run on system nodes. This option suits large and loaded clusters.

The node resource requirements for these options are listed in the Resource requirements section.

Resource requirements

Resource requirements depend on the deployment option and on the solution you choose for the infrastructure tasks (running containers, virtual machines, or both containers and virtual machines).

> **i** The requirements below are given per node and cover only the load of system components. Allocate resources for the user workload in addition to them.

When choosing resources, consider the following:

- The values are calculated for an installation with the `Default` module bundle.
- The values in the “Disk, GB” column include 100 GB for system PVCs (of the `prometheus`, `upmeter` modules and others) on the node’s local disk. If system PVCs are stored in an external storage, this space is not needed on the node.
- The values in the “Disk, GB” column for master nodes do not include the store of the in-cluster cache of the `registry` module. If the cache is enabled (`storage.cache`), each master node additionally needs space for `storage.size` plus the reserve the store leaves free for the node. Preferably, give the store a separate disk of that size mounted at the store directory: on a disk shared with etcd, etcd slows down while the cache is being filled (see “Disk layout and size” and the `registry` module documentation).
- On every worker node, reserve at least 1 CPU and 2 GB of memory for system services (`kubelet`) and system pods.
- For all nodes, use fast disks with performance of at least 400 IOPS.
- An arbiter node in an HA cluster with two master nodes needs at least 2 CPU, 4 GB of memory and 8 GB of disk for etcd. For details, see the Configuring HA mode with two master nodes and an arbiter node section.

For details on resource planning, see the guides:

- Picking resources for a bare metal cluster;

- Disk layout and size;
- Going to production.

Select an option depending on the goals of the Deckhouse Platform deployment:

- For containers
- For virtualization
- For combined workloads

The table lists the Deckhouse Platform resource requirements for running containers, depending on the deployment option:

Cluster configuration	Node type	Number of nodes	vCPU	RAM, GB	Disk, GB
All-in-one cluster where a single node performs both master and worker roles	Master node	1	12	24	350
Cluster with one master node and one or more worker nodes	Master node	1	8	16	150
	Worker node	1+	8	16	250
High-availability cluster with three master nodes	Master node	3	6	12	150
Cluster with role separation: dedicated master, worker, system, frontend and monitoring nodes	Master node	3	6	12	60
	Worker node	1+	2	4	60
	System node	2+	2	4	100
	System node (applied if there are no dedicated monitoring nodes)	2+	6	12	300
	Frontend node	2+	2	4	60
	Monitoring node	2+	4	8	200

The table lists the Deckhouse Platform resource requirements for running virtual machines, depending on the deployment option:

Cluster configuration	Node type	Number of nodes	vCPU	RAM, GB	Disk, GB
All-in-one cluster where a single node performs both master and worker roles	Master node	1	16	32	450
Cluster with one master node and one or more nodes intended for running virtual machines	Master node	1	8	16	150
	Virtual machine node	1+	12	24	350

Cluster configuration	Node type	Number of nodes	vCPU	RAM, GB	Disk, GB
High-availability cluster with three master nodes	Master node	3	8	16	200
Cluster with role separation: dedicated master, virtual-machine, system, frontend and monitoring nodes	Master node	3	6	12	60
	Virtual machine node	1+	4	8	120
	System node	2+	4	8	100
	System node (applied if there are no dedicated monitoring nodes)	2+	8	16	300
	Frontend node	2+	2	4	60
	Monitoring node	2+	4	8	200

Also consider the requirements for running virtual machines.

The table lists the Deckhouse Platform resource requirements for combined workloads (running containers and virtual machines), depending on the deployment option:

Cluster configuration	Node type	Number of nodes	vCPU	RAM, GB	Disk, GB
All-in-one cluster where a single node performs both master and worker roles	Master node	1	16	32	450
Cluster with one master node and one or more nodes intended for running containers or virtual machines	Master node	1	8	16	150
	Worker or virtual machine node	1+	12	24	350
High-availability cluster with three master nodes	Master node	3	8	16	200
Cluster with role separation: dedicated master, worker, virtual-machine, system, frontend and monitoring nodes	Master node	3	6	12	60
	Worker node	1+	2	4	60
	Virtual machine node	1+	4	8	100
	System node	2+	4	8	100
	System node (applied if there are no dedicated monitoring nodes)	2+	8	16	300
	Frontend node	2+	2	4	60

Cluster configuration	Node type	Number of nodes	vCPU	RAM, GB	Disk, GB
	Monitoring node	2+	4	8	200

Also consider the requirements for running virtual machines.

Before the installation, the installer checks that the server (VM) for the first master node has the minimum resources. These values are lower than those in the tables: they are enough for the installation to start, but not for running the cluster under load.

Minimum resources checked by the installer...

The server (VM) for the first master node must have:

- At least 4 CPU cores (at least 2 CPU cores with the `Minimal` module bundle).
- At least 8 GB of RAM (at least 4 GB with the `Minimal` module bundle).
- At least 50 GB of disk space with 400+ IOPS performance.

The other installer checks are described in the Pre-installation checks section.

Node requirements

The servers (VMs) for cluster nodes must meet the following requirements.

Hardware:

- **CPU architecture** — `x86_64` on all nodes.
- **Identical nodes** — all nodes of the same type have the same configuration: the same make and model with the same CPU, memory and storage.
- **Network interfaces** — each node has at least one network interface for the routed network.

Operating system and software:

- An OS from the list of supported ones or a compatible one is installed.
- Linux kernel version 5.8 or newer.
- One of the package managers is installed: `apt`, `apt-get`, `yum`, or `rpm`. For RED OS, `yum` and `which` are installed (they may be missing by default).
- Access to standard OS package repositories for installing dependencies.
- Python and the `sudo` command are available: the installer needs them.
- The hostname is unique within the cluster and meets the Kubernetes requirements: no more than 63 characters, lowercase letters only, hyphens and periods are allowed but not at the beginning or end of the name.
- The node resolves its own hostname, and `localhost` resolves to `127.0.0.1`.
- The system time is correct.
- The node has no container runtime of its own (for example, containerd or Docker) and no Kubernetes components.
- The local `deckhouse` user and group are either absent or have UID and GID `64535`, and the user has no `sudo` privileges. For details, see the The local deckhouse system user section.
- **When using `ContainerdV2`** as the default container runtime on cluster nodes:
  - Support for `CgroupsV2`.
  - Systemd version `244` or newer.
  - Support for the `erofs` kernel module.
  - The Linux kernel version is not in the 6.12.0–6.12.28 or 6.14.0–6.14.6 ranges (these versions are affected by the CVE-2025-37999 vulnerability in EROFS).

The installer checks these requirements before the installation starts. The full list of checks is given in the Pre-installation checks section.

If security policies or security scanners apply to the nodes, take into account the sysctl parameters that the platform sets on nodes and the directories used by the platform.

Network requirements

The following requirements apply to the network between cluster nodes:

- Nodes can reach each other over the network. There are no MTU requirements.
- Each node has a permanent IP address. If a DHCP server assigns the addresses, configure it to assign a fixed address to each node. Changing the IP address of a node is undesirable.
- Master nodes can access external time sources via NTP. The other nodes synchronize time with master nodes, but can also use other time servers (the `ntpServers` parameter).
- If the infrastructure restricts network interaction, the ports from the tables in the Network interaction of the platform components section are open between nodes.

- The sum of the round-trip time (RTT) and jitter between the servers for master nodes and arbiter nodes does not exceed 100 ms, or 50 ms when planning the cluster. For details, see the Requirements for network latency between master nodes section.
- A domain for the platform web interfaces is prepared; it is set in the DNS name template: a wildcard record pointing to the Ingress controller, or an A or CNAME record for every name from the list of reserved DNS names.

Requirements for running virtual machines

⚠ Nodes that run virtual machines have to be physical servers (bare metal). Installation on virtual machines is allowed for demonstration purposes only, requires nested virtualization to be enabled, and isn't covered by technical support.

Nodes and infrastructure for running VMs have to meet additional requirements. Take them into account when planning the cluster, because some settings can't be changed after the installation:

- A CPU supporting Intel-VT (VMX) or AMD-V (SVM) instructions, with hardware virtualization enabled in the BIOS/UEFI settings.
- The same Linux kernel version on all cluster nodes.
- Astra Linux nodes require Astra Linux version 1.8.3 or later: earlier versions contain a bug that interferes with virtualization.
- One or several supported storages in the cluster. Also set the default StorageClass.
- The subnets that VMs will get IP addresses from (set in the `virtualMachineCIDRs` parameter) must not overlap the pod (`podSubnetCIDR`) and service (`serviceSubnetCIDR`) address spaces.

A cluster with virtualization is designed to run up to `1000` nodes and up to `50000` virtual machines.

Access to the infrastructure

The installation requires the following access:

- Access to the Deckhouse Platform container image registry: the public one (`registry.deckhouse.io` or `registry.deckhouse.ru`) or its mirror.
- For an installation on bare metal (including a hybrid cluster) or in an unsupported cloud — access to nodes over SSH with a key.
- For an installation of a hybrid cluster — network connectivity between the sites according to the general network requirements for hybrid clusters.
- For an installation in a supported cloud — quotas for creating resources and the cloud infrastructure access parameters. They depend on the provider and are described in the Infrastructure integration section.
- For an installation into an existing Kubernetes cluster — access to the Kubernetes API of the cluster from the installer host with administrator rights (kubeconfig). Installation directly from a master node is not supported.
- On the host that runs the installer — a container runtime for running the installer container, access to the container image registry, and network access to the master node server on port `22/TCP`.

The requirements for the bastion host and the image registry in an air-gapped environment are listed in the Installing Deckhouse Platform in an air-gapped environment guide.

Additional resources

- Getting started
- Disk layout and size
- Going to production
- Installing Deckhouse Platform in a private environment
- Supported Kubernetes and OS versions
- Network interaction of the platform components

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Installation

> **i** Step-by-step installation instructions are available in the Getting started section.

This page provides an overview of installing Deckhouse Platform.

Installation methods

You can install Deckhouse Platform using a CLI installer, which is available as a container image and based on the dhctl utility.

Installation options

You can install Deckhouse Platform in the following ways::

- **In a supported cloud.** The installer automatically creates and configures all required resources (including virtual machines, network objects, etc.), deploys a Kubernetes cluster, and installs Deckhouse Platform. A full list of supported cloud providers is available in the Integration with IaaS section.
- **On bare-metal servers (including hybrid clusters) or in unsupported clouds.** The installer configures the servers or virtual machines specified in the configuration, deploys a Kubernetes cluster, and installs Deckhouse Platform. Step-by-step instructions for bare metal are available in Getting started → Deckhouse Platform for bare metal.
- **In an existing Kubernetes cluster.** The installer deploys Deckhouse Platform and integrates it with the current infrastructure. Step-by-step instructions for an existing cluster are available in Getting started → Deckhouse Platform in existing cluster.

Installation requirements

Before starting the installation, review the Deckhouse Platform deployment scenarios and installation requirements in the “System Requirements” section.

Preparing the Configuration

Before installation, you need to prepare the installation configuration file and, if needed, a post-bootstrap script.

Installation configuration file

The installation configuration file is a set of YAML documents that contains Deckhouse Platform settings and manifests for cluster objects and resources to be created after installation. The configuration file is used by the CLI installer and is passed via the `--config` parameter (see below).

Required and optional objects/resources that may be needed in the installation configuration file:

1. InitConfiguration (**required**): Initial configuration parameters necessary to start Deckhouse Platform.

   > **i** Starting with Deckhouse Platform 1.75, use the ModuleConfig `deckhouse` to configure access to the Deckhouse Platform container registry. Configuring access with InitConfiguration (via `imagesRepo`, `registryDockerCfg`, `registryScheme`, and `registryCA` parameters) is considered a legacy method.

2. ClusterConfiguration: General cluster parameters, such as Kubernetes (control plane components) version, network settings, CRI parameters, etc. **Required**, except when Deckhouse Platform is installed into an already existing Kubernetes cluster.
3. StaticClusterConfiguration: Parameters for a cluster deployed on bare-metal servers (including hybrid clusters) or in unsupported clouds. **Required**, except when Deckhouse Platform is installed into an already existing Kubernetes cluster.

   To add worker node groups (the NodeGroup object), you may also need StaticInstance and SSHCredentials.
4. `<PROVIDER>ClusterConfiguration`: Parameters for integration with a cloud provider. **Required** when integrating Deckhouse Platform with a supported cloud infrastructure.

> **i** The DVPClusterConfiguration, YandexClusterConfiguration, and ZvirtClusterConfiguration resources are deprecated. The parameters for integration with built-in virtualization, Yandex Cloud, and zVirt are set in the ModuleConfig of the `cloud-provider-dvp`, `cloud-provider-yandex`, and `cloud-provider-zvirt` modules and in the related resources. These resources are described in Layouts and configuration in built-in virtualization, Layouts and configuration in Yandex Cloud, and Layouts and configuration in zVirt.

Examples of resources configuring integration with a cloud provider:

- AWSClusterConfiguration: Amazon Web Services
- AzureClusterConfiguration: Microsoft Azure
- ModuleConfig `cloud-provider-dvp`: Deckhouse Virtualization Platform (together with the `d8-credentials` Secret and DVPInstanceClass / NodeGroup resources)
- GCPClusterConfiguration: Google Cloud Platform
- HuaweiCloudClusterConfiguration: Huawei Cloud
- OpenStackClusterConfiguration: OpenStack, OVHcloud, Selectel, VK Cloud
- VsphereClusterConfiguration: VMware vSphere
- VCDClusterConfiguration: VMware Cloud Director
- ModuleConfig `cloud-provider-yandex`: Yandex Cloud (together with the `d8-credentials` Secret and YandexInstanceClass / NodeGroup resources)
- ModuleConfig `cloud-provider-zvirt`: zVirt (together with the `d8-credentials` Secret and ZvirtInstanceClass / NodeGroup resources)

To add cloud nodes, you also need `<PROVIDER>InstanceClass` objects (for example YandexInstanceClass for Yandex Cloud) that describe VM configuration in the node group (the NodeGroup object).

5. Deckhouse Platform module configurations.

   Each module is configured (and can be enabled or disabled) with its own ModuleConfig (for example, ModuleConfig `user-authn` for the `user-authn` module). Parameters that are allowed in the ModuleConfig object are described in the respective module documentation under “Configuration” (for example, configuration of the `user-authn` module).

   A list of all Deckhouse Platform modules is available in the Modules section.

   Some modules may be enabled and preconfigured automatically depending on the selected installation option and cluster configuration (for example, modules that provide control plane and networking functionality).

   Modules often configured during installation:

   - `global`: Global Deckhouse Platform settings for parameters used by default by all modules and components (DNS name template, StorageClass, module component placement settings, etc.).
   - `deckhouse`: Container registry access settings, the desired release channel, and other parameters.
   - `user-authn`: Unified authentication.
   - `cni-cilium`: Cluster networking (for example, used when installing Deckhouse Platform on bare metal or in an air-gapped environment);
   - `virtualization`: Running virtual machines in the cluster, enabled during the installation or later.

   If the cluster is created with nodes dedicated to specific workload types (for example, system or monitoring nodes), it is recommended to explicitly set the `nodeSelector` parameter in module configurations that use persistent storage volumes (for example, in the `nodeSelector` parameter of the `prometheus` ModuleConfig for the `prometheus` module).

6. IngressNginxController: Parameters of the HTTP/HTTPS load balancer (Ingress controller).
7. NodeGroup: Node group parameters. Required to add worker nodes.
8. Objects for authentication and authorization such as ClusterAuthorizationRule, AuthorizationRule, User, Group, and DexProvider.

   See authentication and authorization documentation for details.

An example of the installation config...

- Configuration applicable since Deckhouse Platform 1.75
- Legacy configuration

In this example, access to the Deckhouse Platform container registry is configured using ModuleConfig

`deckhouse`.

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
---
# control-plane-manager module settings.
# https://deckhouse.io/modules/control-plane-manager/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "Default"
    network:
      podSubnetCIDR: 10.111.0.0/16
      serviceSubnetCIDR: 10.222.0.0/16
      clusterDomain: cluster.local
    apiserver:
      publishAPI:
        ingress:
          enabled: true
          https:
            mode: Global
            global:
              kubeconfigGeneratorMasterCA: ""
---
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
vNetCIDR: 10.241.0.0/16
subnetCIDR: 10.241.0.0/24
masterNodeGroup:
  replicas: 3
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
    enableExternalIP: true
provider:
  subscriptionId: <SUBSCRIPTION_ID>
  clientId: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
  tenantId: <TENANT_ID>
  location: westeurope
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
  settings:
    releaseChannel: Stable
    bundle: Default
    logLevel: Info
```

```
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: test-registry.io/some/path
        scheme: HTTPS
        username: <username>
        password: <password>
        ca: <CA>
  version: 1
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  settings:
    modules:
      publicDomainTemplate: "%s.k8s.example.com"
  version: 2
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: DoNotNeed
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 1
  enabled: true
  settings: {}
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Specify if you plan to use dedicated nodes for monitoring.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
```

```
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
  - kind: User
    name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1XaOGg2'
```

In this example, access to the Deckhouse Platform container registry is configured using InitConfiguration.

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
---
# control-plane-manager module settings.
# https://deckhouse.io/modules/control-plane-manager/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "Default"
    network:
```

```
    podSubnetCIDR: 10.111.0.0/16
    serviceSubnetCIDR: 10.222.0.0/16
    clusterDomain: cluster.local
  apiserver:
    publishAPI:
      ingress:
        enabled: true
        https:
          mode: Global
          global:
            kubeconfigGeneratorMasterCA: ""
---
apiVersion: deckhouse.io/v1
kind: InitConfiguration
deckhouse:
  imagesRepo: registry.deckhouse.io/deckhouse/ee
  registryDockerCfg: eyJhdXRocyI6IHsgInJlZ2zzzmRlY2tob3Vxxcxxxc5ydSI6IsssfX0K
---
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
vNetCIDR: 10.241.0.0/16
subnetCIDR: 10.241.0.0/24
masterNodeGroup:
  replicas: 3
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
    enableExternalIP: true
provider:
  subscriptionId: <SUBSCRIPTION_ID>
  clientId: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
  tenantId: <TENANT_ID>
  location: westeurope
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
  settings:
    releaseChannel: Stable
    bundle: Default
    logLevel: Info
  version: 1
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  settings:
    modules:
      publicDomainTemplate: "%s.k8s.example.com"
  version: 2
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
```

```
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: DoNotNeed
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 1
  enabled: true
  settings: {}
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Specify if you plan to use dedicated nodes for monitoring.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
```

```
metadata:
  name: admin
spec:
  subjects:
  - kind: User
    name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1XaOGg2'
```

Post-bootstrap script

The installer allows you to run a custom script on one of the master nodes after installation (post-bootstrap script). This script can be used for:

- Additional cluster configuration
- Collecting diagnostic information
- Integrating with external systems or other tasks

The path to the post-bootstrap script can be specified using the `--post-bootstrap-script-path` parameter when running the CLI installer.

Example: a script that retrieves the IP address of the load balancer...

This sample script retrieves the IP address of the load balancer after Deckhouse Platform is installed:

```
#!/usr/bin/env bash

set -e
set -o pipefail


INGRESS_NAME="nginx"


echo_err() { echo "$@" 1>&2; }

# Declare the variable.
lb_ip=""

# Get the load balancer IP address.
for i in {0..100}
do
  if lb_ip="$(kubectl -n d8-ingress-nginx get svc "${INGRESS_NAME}-load-balancer" -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')"; then
    if [ -n "$lb_ip" ]; then
      break
    fi
  fi

  lb_ip=""

  sleep 5
done

if [ -n "$lb_ip" ]; then
  echo_err "The load balancer external IP: $lb_ip"
else
  echo_err "Could not get the external IP of the load balancer"
  exit 1
fi

outContent="{\"frontend_ips\":[\"$lb_ip\"]}"

if [ -z "$OUTPUT" ]; then
  echo_err "The OUTPUT env is empty. The result was not saved to the output file."
else
  echo "$outContent" > "$OUTPUT"
fi
```

Installing

> **i** When installing a commercial edition of Deckhouse Platform from the official container registry `registry.deckhouse.io`, you must first log in with your license key:
>
> ```
> docker login -u license-token registry.deckhouse.io
> ```

The command to run the installer container from the public Deckhouse container registry:

```
docker run --pull=always -it [<MOUNT_OPTIONS>]
registry.deckhouse.io/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

Where:

1. `<DECKHOUSE_REVISION>`: Deckhouse Platform edition. For example, `ee` for Enterprise Edition, `ce` for Community Edition, etc.
2. `<MOUNT_OPTIONS>`: Parameters for mounting files into the installer container, such as:
   - SSH access keys

    - Configuration file
    - Resource file, etc.
3. `<RELEASE CHANNEL>`: Release channel in kebab-case format:
    - `alpha`: Alpha channel
    - `beta`: Beta channel
    - `early-access`: Early Access channel
    - `stable`: Stable channel
    - `rock-solid`: Rock Solid channel

Here is an example of a command to run the Deckhouse Platform Community Edition/Open installer container from the Stable release channel:

```
docker run -it --pull=always \
  -v "$PWD/config.yaml:/config.yaml" \
  -v "$PWD/dhctl-tmp:/tmp/dhctl" \
  -v "$HOME/.ssh/:/tmp/.ssh/" registry.deckhouse.io/deckhouse/ce/install:stable bash
```

Deckhouse Platform installation is performed within the installer container using the `dhctl` command:

- To start the installation of Deckhouse Platform with the deployment of a new cluster (for all cases except installing into an existing cluster), use the command `dhctl bootstrap`.
- To install Deckhouse Platform into an already existing cluster, use the command `dhctl bootstrap-phase install-deckhouse`.

> **i** To learn more about the available parameters, run `dhctl bootstrap -h`.

Example of running a Deckhouse Platform installation with cloud cluster deployment:

```
dhctl bootstrap \
  --ssh-user=<SSH_USER> --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
  --config=/config.yml
```

Where:

- `/config.yml`: Installation configuration file.
- `<SSH USER>`: Username for SSH connection to the server.
- `--ssh-agent-private-keys`: Private SSH key file for SSH connection.
- `<SSH_PRIVATE_KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

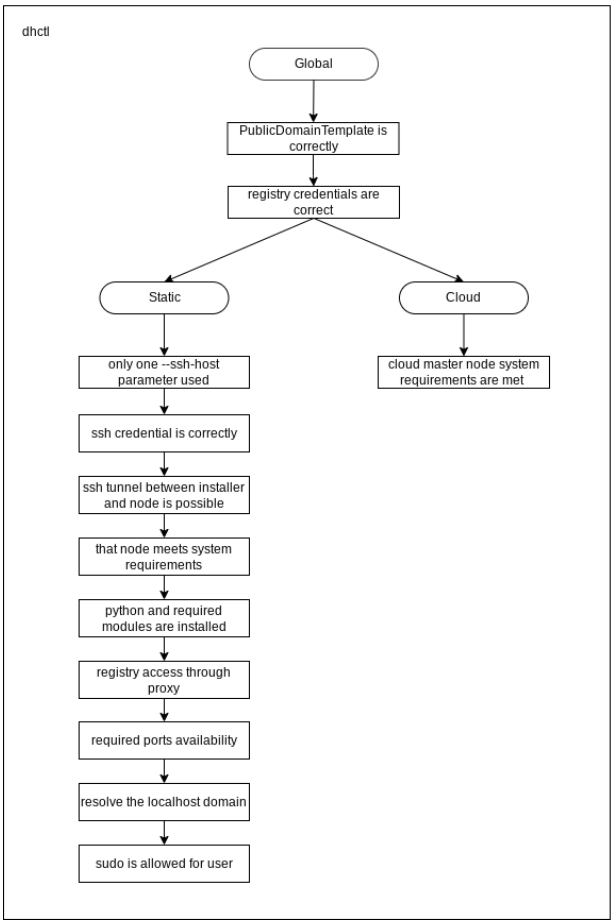
If the server accepts only keys signed by an SSH certificate authority, put the certificate next to the private key as `<SSH_PRIVATE_KEY_FILE>-cert.pub`, for example `id_ed25519-cert.pub`. dhctl uses it automatically.

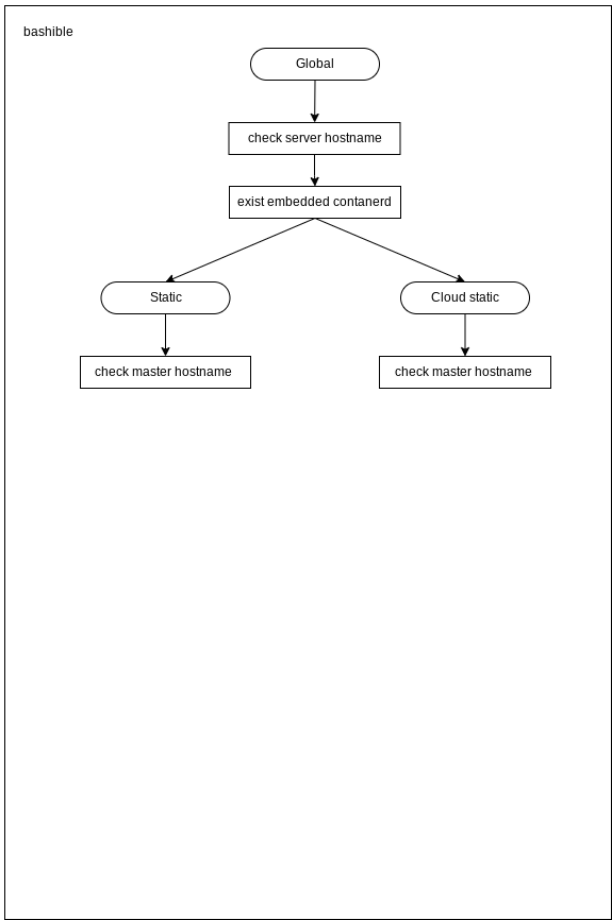
Pre-installation checks

> **i** Starting with version 1.74, Deckhouse Platform modules are installed as images in the EROFS format that are mounted read-only, which protects them from being modified after installation. This mechanism is enabled automatically if the `erofs` filesystem is registered in the kernel on the node running the Deckhouse Platform controller (a master node by default). Deckhouse Platform loads this kernel module only on nodes with containerd v2, so if master nodes use containerd v1, the operating system has to load `erofs` on its own. Otherwise, Deckhouse Platform will install modules the regular way, without protecting their integrity and without a dedicated alert. For details, refer to “Integrity protection of Deckhouse Platform modules”.

Diagram of checks performed by the installer before installation...

Preflight checks





List of checks performed by the installer before starting Deckhouse Platform installation:

1. General checks:
   - The values of the parameters `publicDomainTemplate` and `clusterDomain` (or the deprecated `ClusterConfiguration.clusterDomain`) do not match.
   - The authentication data for the container registry specified in the installation configuration is correct.
   - The host name meets the following requirements:
     - The length does not exceed 63 characters.
     - It consists only of lowercase letters.
     - It does not contain special characters (hyphens `-` and periods `.` are allowed, but they cannot be at the beginning or end of the name).
   - The server (VM) has a supported container runtime (`containerd`) installed.
   - The host name is unique within the cluster.
   - The server's system time is correct.
   - The address spaces for Pods (`podSubnetCIDR`) and services (`serviceSubnetCIDR`) do not intersect.
2. Checks for static and hybrid cluster installation:
   - Only one `--ssh-host` parameter is specified. For static cluster configuration, only one IP address can be provided for configuring the first master node.
   - SSH connection is possible using the specified authentication data.
   - SSH tunneling to the master node server (or VM) is possible.
   - The server (VM) selected for the master node installation has the minimum resources listed in the Resource requirements section. The same section lists the node resources for each deployment option.
   - Python (Python 3 or Python 2) must be available on the server (VM) intended for the master node. The standard Python modules used by the installer must also be available:
     - For HTTP request handling: `urllib.request` (Python 3) or `urllib2` (Python 2)
     - For HTTP error handling: `urllib.error` (Python 3) or `urllib2` (Python 2)
     - For working with configuration files: `configparser` (Python 3) or `ConfigParser` (Python 2)
     - For HTTP server: `http.server` (Python 3) or `SimpleHTTPServer` (Python 2)
     - For TCP server: `http.server` (Python 3) or `SocketServer` (Python 2)
   - The container registry is accessible through a proxy (if proxy settings are specified in the installation configuration).
   - On the server (VM) intended for the master node and on the host from which the installer is launched, port `22/TCP` (network access from a personal computer) must be accessible, as it

    is required for the installation process. For details, see the Network interaction of the platform components section.
    - DNS must resolve `localhost` to IP address `127.0.0.1`.
    - The user has `sudo` privileges on the server (VM).
    - Required ports for the installation must be open:
      - Port `22/TCP` between the host running the installer and the server.
      - No port conflicts with those used by the installation process.
    - The server (VM) has the correct time.
    - The server (VM) has no conflicts with the local `deckhouse` system user and group: the user and group are either absent, or the user has UID and primary GID `64535`, the group has GID `64535`, the user has no `sudo` privileges, and UID/GID `64535` are not assigned to other accounts. For details, see The local `deckhouse` system user.
    - The address spaces for Pods (`podSubnetCIDR`), services (`serviceSubnetCIDR`), and internal network (`internalNetworkCIDRs`) do not intersect.
3. Checks for cloud cluster installation:
    - The configuration of the virtual machine for the master node has the minimum resources listed in the Resource requirements section.
    - The cloud provider API is accessible from the cluster nodes.
    - The configuration for Yandex Cloud with NAT Instance is verified.

List of preflight skip flags...

To skip a specific check, use the `--preflight-skip-check` flag and pass the preflight check name as its argument. The flag can be specified multiple times.

`dhctl preflight list` prints every check, what it applies to and what it asserts.

- `--preflight-skip-all-checks`: Skip all preflight checks.

Configuration checks (the `PreInfraPreflights` phase):

- `--preflight-skip-check=registry-credentials`: Skip the check for registry access credentials.
- `--preflight-skip-check=registry-reachable`: Skip the check that the registry answers from the installer host.
- `--preflight-skip-check=deckhouse-image-available`: Skip the check that the Deckhouse image of this version is in the registry.
- `--preflight-skip-check=registry-required-images`: Skip the check that the registry holds the images the release is made of, and not just the tag that names it.
- `--preflight-skip-check=dhctl-edition`: Skip the check that the installer edition matches the Deckhouse image edition.
- `--preflight-skip-check=dhctl-version`: Skip the check that the installer version matches the Deckhouse image version.
- `--preflight-skip-check=bastion-availability`: Skip the bastion host availability check.
- `--preflight-skip-check=cloud-disk-name-length`: Skip the check that the cluster prefix does not make generated disk names too long.
- `--preflight-skip-check=cloud-master-system-requirements`: Skip the check that the master node instance class meets the minimum requirements.
- `--preflight-skip-check=instance-class-provider`: Skip the check that InstanceClass resources match the cloud provider.
- `--preflight-skip-check=cloud-node-network-cidr-intersection`: Skip the check that the cluster subnets do not overlap the cloud network the nodes are created on.
- `--preflight-skip-check=cloud-ssh-key-matches-public-key`: Skip the check that a private key dhctl holds matches the `sshPublicKey` the cloud will install.
- `--preflight-skip-check=dynamix-storage-policy`: Skip the check that Basis Dynamix is version 4.6 or newer and that every storage policy named in DynamixClusterConfiguration is `ENABLED` and available to the account.
- `--preflight-skip-check=static-instances-ip-duplication`: Skip the check for duplicate StaticInstance addresses.
- `--preflight-skip-check=immutable-installer-images`: Skip the check that the installer image carries the requested Kubernetes control plane.
- `--preflight-skip-check=immutable-kubeconfig-kept`: Skip the check that the admin kubeconfig is written outside the directory dhctl cleans up.
- `--preflight-skip-check=immutable-kubeconfig-out`: Skip the check that the admin kubeconfig has a destination.
- `--preflight-skip-check=immutable-machines-availability`: Skip the check that the machines named with `--master-host` answer.
- `--preflight-skip-check=immutable-post-bootstrap-script`: Skip the check that no post-bootstrap script is requested.
- `--preflight-skip-check=immutable-signature-mode`: Skip the check that the control-plane signature mode is off.
- `--preflight-skip-check=immutable-supported-provider`: Skip the check that the platform is supported for an immutable master.

Node checks (the `PostInfraPreflights` phase):

- `--preflight-skip-check=static-single-ssh-host`: Skip the check for the number of specified SSH hosts.
- `--preflight-skip-check=ssh-connectivity`: Skip the check that the node answers on its SSH port.
- `--preflight-skip-check=ssh-credential`: Skip the check that the SSH user can log in to the node.
- `--preflight-skip-check=static-ssh-tunnel`: Skip the SSH forwarding check.
- `--preflight-skip-check=static-instances-ssh-access`: Skip the check for SSH access to StaticInstances.
- `--preflight-skip-check=sudo-installed`: Skip the check that `sudo` is installed on the node.
- `--preflight-skip-check=sudo-allowed`: Skip the check for `sudo` privileges.
- `--preflight-skip-check=deckhouse-user`: Skip checks of the `deckhouse` user and group, including their UID/GID, conflicts with UID/GID `64535`, and whether the `deckhouse` user has `sudo` privileges.
- `--preflight-skip-check=node-system-requirements`: Skip the check for meeting system requirements.
- `--preflight-skip-check=python-modules`: Skip the check for Python installation.
- `--preflight-skip-check=ports-availability`: Skip the check for the availability of required ports.
- `--preflight-skip-check=resolve-localhost`: Skip the `localhost` resolution check.
- `--preflight-skip-check=time-drift`: Skip the time drift check.
- `--preflight-skip-check=node-hostname`: Skip the check that the node hostname is one Kubernetes accepts as a node name.
- `--preflight-skip-check=node-os-supported`: Skip the check that the node runs an operating system Deckhouse can configure.
- `--preflight-skip-check=node-xfs-ftype`: Skip the check for an XFS filesystem formatted without `d type`.
- `--preflight-skip-check=node-resolve-hostname`: Skip the check that the node resolves its own hostname.
- `--preflight-skip-check=node-leftovers`: Skip the check that the node carries no container runtime or Kubernetes of its own.
- `--preflight-skip-check=node-cri-requirements`: Skip the check that the node meets what `ContainerdV2` needs (kernel, systemd, cgroup v2, `erofs`).
- `--preflight-skip-check=node-kernel-modules`: Skip the check that the kernel modules Deckhouse loads can be loaded on the node.
- `--preflight-skip-check=node-selinux-tools`: Skip the check that a node with SELinux enforcing has the tools to install the Deckhouse policy.
- `--preflight-skip-check=node-disk-space`: Skip the check that the node has the disk Deckhouse needs.
- `--preflight-skip-check=static-free-disk-space`: Skip the check that the node has enough free disk for the bootstrap.
- `--preflight-skip-check=node-internal-network`: Skip the check that the node has an address inside `internalNetworkCIDRs`.
- `--preflight-skip-check=host-network-cidr-intersection`: Skip the check for intersection between the cluster CIDRs and the host networks.
- `--preflight-skip-check=registry-access-through-proxy`: Skip the check for accessing the registry through a proxy server.
- `--preflight-skip-check=cloud-api-accessibility`: Skip the Cloud API accessibility check.
- `--preflight-skip-check=cloud-api-from-installer`: Skip the check that the installer itself reaches the cloud provider API.
- `--preflight-skip-check=network-single-source`: Skip the check that the cluster network parameters are declared in only one of `ClusterConfiguration` or the `control-plane-manager` ModuleConfig.
- `--preflight-skip-check=cloud-kube-data-device`: Skip the check that the disk the provider attached for Kubernetes data is on the master.
- `--preflight-skip-check=registry-access-from-master`: Skip the check that the master node itself can reach the registry.
- `--preflight-skip-check=immutable-api-reachable`: Skip the check that the API port of an immutable first master answers.

The cluster subnets and `publicDomainTemplate` are not preflight checks: they read only the configuration, so they are validated when it is loaded — by `dhctl config`, by converge and by bootstrap alike. No flag can skip them.

The names `cidr-intersection`, `static-cidr-intersection` and `public-domain-template` are accepted for compatibility but do not affect check execution.

When a check name is specified with `--preflight-skip-check`, both checks corresponding to that name are skipped: `registry-credentials` skips `registry-reachable`, `dhctl-edition` skips `deckhouse-image-available`, `sudo-allowed` skips `sudo-installed`, and `ssh-credential` skips `ssh-connectivity`.

Checks that concern the machine itself are named `node-*`, while checks that concern access to it are named `ssh-*`. Both run on a cloud master as well as on a static node. Checks with the `static-` prefix apply only to a static cluster. Renamed check names are accepted by `--preflight-skip-check`.

The node checks — `sudo-installed`, `sudo-allowed`, `python-modules`, `resolve-localhost`, `time-drift`, `node-hostname`, `node-disk-space`, `node-leftovers`, `node-cri-requirements`, `node-internal-network`, `node-system-requirements` and `host-network-cidr-intersection` — also run on a cloud master as well as on a static node. Their names are the same on both paths.

Two more flags control check execution:

- `--preflight-fail-fast`: Stop at the first failed check instead of running the remaining checks in the phase.
- `--preflight-no-cache`: Run all checks again without using results saved from the previous run.

Example of using a preflight skip flag:

```
dhctl bootstrap \
--ssh-user=<SSH_USER> --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
--config=/config.yml \
--preflight-skip-all-checks
```

> **i** Replace `<SSH_PRIVATE_KEY_FILE>` here with the name of your private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

The local `deckhouse` system user

Deckhouse Platform components use UID/GID `64535` to run unprivileged processes and access their files on the node. The local `deckhouse` user and group reserve these identifiers for Deckhouse Platform and let the OS display the `deckhouse` name instead of numeric UID/GID values in tools such as `ps`, `top`, and `ls`. Processes owned by `deckhouse` in `ps` or `top` output are expected: they belong to Deckhouse Platform components, not to user sessions.

If the account does not exist, bashible creates it with the `/sbin/nologin` shell, without a password or `sudo` privileges. `/etc/passwd` lists the system default home directory for the user (usually `/home/deckhouse`). Bashible creates this directory separately as the base directory for the home directories of users defined by `NodeUser` resources; the `deckhouse` system account itself is not intended for login or storing user files. The account must not be provided through AD or LDAP.

The `deckhouse` name and UID/GID `64535` are reserved for Deckhouse Platform and are not configurable. Do not delete or rename the user or group, change their UID/GID, or grant the user `sudo` privileges. The preflight check accepts an absent user and group, or an existing pair where the user has UID and primary GID `64535`, the group has GID `64535`, and the user has no `sudo` privileges. Installation fails if the name is used with different IDs or UID/GID `64535` are assigned to other accounts.

The `--preflight-skip-check=deckhouse-user` flag disables the check but does not resolve the conflict. If the `deckhouse` user or group exists with different IDs, bashible does not modify it and logs a warning. If UID/GID `64535` are assigned to another account, bashible cannot create the `deckhouse` user or group. Nor does bashible revoke `sudo` privileges from an existing user.

If the conflict was left by a previous Deckhouse Platform installation on a static node, perform manual cleanup of the static node. This is a destructive operation: the cleanup script removes the services, configuration, and local Kubernetes, containerd, and etcd data created by Deckhouse Platform, as well as Deckhouse Platform accounts, and then reboots the server. Run the cleanup only on a node prepared for redeployment. In all other cases, resolve the conflict manually or recreate the node.

Aborting the installation

If the installation was interrupted or issues occurred during the installation process in a supported cloud, there might be leftover resources created during the installation. To remove them, run the following command within the installer container:

```
dhctl bootstrap-phase abort
```

> **⚠** The configuration file provided through the `--config` parameter when running the installer must be the same that was used during the initial installation.

Installing virtualization

Virtualization is enabled by the `virtualization` ModuleConfig resource and deploys its components to the `d8-virtualization` namespace. You can either include the resource in the installation configuration file or apply it in a ready cluster.

> ⚠ Enabling virtualization restarts kubelet, containerd, and the network module agents on every node that is expected to run virtual machines. This is required to set up connectivity between containerd and the DVCR image storage.

A minimal configuration defines the image storage size and the subnets that virtual machines get addresses from:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: virtualization
spec:
  enabled: true
  version: 1
  settings:
    dvcr:
      storage:
        type: PersistentVolumeClaim
        persistentVolumeClaim:
          size: 50G
    virtualMachineCIDRs:
      - 10.66.10.0/24
```

The other parameters are covered in the module settings reference.

To check if virtualization is ready, run the following command:

```
d8 k get modules virtualization
```

Example output:

```
NAME             WEIGHT   SOURCE      PHASE   ENABLED   READY
virtualization   900      deckhouse   Ready   True      True
```

The module phase should be `Ready`.

Placement of virtualization components across nodes

How components are distributed across nodes depends on the cluster configuration. A cluster may have only master nodes running both control plane components and workloads; master and worker nodes; master, system, and worker nodes; or other combinations. Worker nodes here are the nodes without restrictions (taints) that would prevent regular workloads from running.

Deckhouse Platform distributes the components by priority, and if the cluster has a suitable node type, the component lands on it. What each component is responsible for is described in Virtualization subsystem.

Component name	Node group	Comment
`virt-operator-*`	system/master	
`virt-api-*`	master	
`virt-controller-*`	system/worker	
`virt-handler-*`	Nodes with KVM	
`virtualization-api-*`	master	

Component name	Node group	Comment
`virtualization-controller-*`	master	
`dvcr-*`	system	Storage has to be available on the node. With no system nodes, the component lands on a worker node.
`virtualization-audit-*`	master	Available in commercial Deckhouse Platform editions.
`virtualization-dra-*`	Dedicated nodes	Available in commercial Deckhouse Platform editions.
`vm-route-forge-*`	All cluster nodes	

The `virt-handler-*` component runs only on the nodes where Deckhouse Platform detected KVM support and set the `virtualization.deckhouse.io/kvm-enabled` label, and the `virtualization-dra-*` component runs only on nodes labeled with `virtualization.deckhouse.io/usbip`.

The components that create and upload virtual machine images and disks run only while that work is in progress:

Component name	Node group	Comment
`d8v-vi-importer-*`, `d8v-cvi-importer-*`	system/worker	Uploads an image from an external source or another resource to the image storage.
`d8v-vi-uploader-*`, `d8v-cvi-uploader-*`	system/worker	Accepts the file that you upload from the CLI or the web interface.
`d8v-vd-pvc-importer-*`, `d8v-vi-pvc-importer-*`	system/worker	Moves an image from the image storage to a disk volume.
`d8v-pvc-pvc-source-importer-*`	system/worker	Serves the data of the source volume over the network when a disk is cloned.
`d8v-pvc-pvc-target-importer-*`	system/worker	Receives the data on the target volume when a disk is cloned.
`d8v-vi-bounder-*`	system/worker	Keeps the volume on the required node while an image is created on it.

Cluster with taints on all nodes

Sometimes taints are configured on every cluster node, which is how an administrator explicitly controls where workloads land. In such a configuration, keep the following in mind:

1. When creating a VirtualDisk, pay attention to the `volumeBindingMode` of the StorageClass. With `Immediate`, the PersistentVolume is created right after the disk is created, before the virtual machine is scheduled. Make sure the storage can create volumes on the nodes allowed for virtual machines by the placement parameters, including `nodeSelector`, `tolerations`, and the settings in the `spec` of the virtual machine or the VirtualMachineClass. Otherwise, the disk may end up on a node where the virtual machine can’t start. With `WaitForFirstConsumer`, the volume is created on the node the virtual machine is scheduled to, and the problem doesn’t occur.
2. VirtualImage and ClusterVirtualImage rely on the temporary components from the table above. They have a toleration for the `dedicated.deckhouse.io=system` taint.
3. The cluster has to have a `system` NodeGroup, or the administrator adds the `dedicated.deckhouse.io=system` taint to selected nodes without creating a NodeGroup. Without such nodes, these components won’t be scheduled, and images won’t reach the `Ready` phase.

Air-gapped environment, working via proxy and using external registries

> **i** For more details on installing and updating Deckhouse Platform in an air-gapped environment, see the “Installing Deckhouse Platform in an air-gapped environment” and “Updating Deckhouse Platform in an air-gapped environment” guides.

Installing from an external (third-party) registry

Deckhouse Platform can be installed from an external container registry or via a proxy registry inside an air-gapped environment.

> **⚠** Deckhouse Platform supports Basic and Bearer token authentication schemes for container registries (Basic is tried first; if it fails, Bearer is used).
>
> If a reverse proxy is placed in front of the registry, it must correctly forward the Registry API v2 header `Docker-Distribution-API-Version: registry/2.0`. Otherwise the Basic check may fail, and the subsequent Bearer attempt may fail with the error `couldn't find bearer realm parameter`.
>
> The following container registries were verified and are guaranteed to work: Nexus, Harbor, Artifactory, CNCF Distribution, Quay.
>
> When working with external registries, do not use an administrator account to access them from Deckhouse Platform. Create a dedicated read-only account limited to the required repository in the registry. Refer to an example of creating such an account.

There are several options for configuring access to external container registries during cluster installation:

- Starting from Deckhouse Platform version 1.75 — using the `deckhouse` ModuleConfig.
- Prior to Deckhouse Platform version 1.75 — using InitConfiguration (considered a legacy method, refer to the example below).

To configure access using the `deckhouse` ModuleConfig, specify the external registry access parameters in the `settings.registry` section.

Specify the parameters for accessing a third-party registry in the `settings.registry` section of the ModuleConfig `deckhouse`.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        imagesRepo: test-registry.io/some/path
        scheme: HTTPS
        username: <username>
        password: <password>
        ca: <CA>
```

Configuration of an external registry using InitConfiguration **(legacy method)**

Set the following parameters in `InitConfiguration`:

- `imagesRepo: <PROXY_REGISTRY>/<DECKHOUSE_REPO_PATH>/ee`: Path to the Deckhouse Platform EE image in an external registry. Example: `imagesRepo: registry.deckhouse.io/deckhouse/ee`.
- `registryDockerCfg: <BASE64>`: Base64-encoded access credentials to the external registry.

If anonymous access is allowed to Deckhouse Platform images in the external registry, the

`registryDockerCfg` should look as follows:

```
{"auths": { "<PROXY_REGISTRY>": {}}}
```

The provided value must be Base64-encoded.

If authentication is required to access Deckhouse Platform images in the external registry, the `registryDockerCfg` should look as follows:

```
{"auths": { "<PROXY_REGISTRY>": {"username":"<PROXY_USERNAME>","password":"
<PROXY_PASSWORD>","auth":"<AUTH_BASE64>"}}}
```

Where:

- `<PROXY_USERNAME>`: Username for authenticating to `<PROXY_REGISTRY>`
- `<PROXY_PASSWORD>`: Password for authenticating to `<PROXY_REGISTRY>`
- `<PROXY_REGISTRY>`: Address of the external registry in the `<HOSTNAME>[:PORT]` format
- `<AUTH_BASE64>`: A Base64-encoded string of `<PROXY_USERNAME>:<PROXY_PASSWORD>`

The final value for `registryDockerCfg` must also be Base64-encoded.

You can use the following script to generate the `registryDockerCfg`:

```
declare MYUSER='<PROXY_USERNAME>'
declare MYPASSWORD='<PROXY_PASSWORD>'
declare MYREGISTRY='<PROXY_REGISTRY>'

MYAUTH=$(echo -n "$MYUSER:$MYPASSWORD" | base64 -w0)
MYRESULTSTRING=$(echo -n "{\"auths\":{\"$MYREGISTRY\":
{\"username\":\"$MYUSER\",\"password\":\"$MYPASSWORD\",\"auth\":\"$MYAUTH\"}}}" |
base64 -w0)

echo "$MYRESULTSTRING"
```

To support non-standard configurations of external registries, InitConfiguration provides two additional parameters:

- `registryCA`: Root certificate to validate the registry's certificate (used if the registry uses self-signed certificates).
- `registryScheme`: Protocol used to access the registry (`HTTP` or `HTTPS`). The default is `HTTPS`.

Nexus configuration notes

⚠ When interacting with a `docker`-type repository in Nexus (for example, using `docker pull` or `docker push` commands), you must specify the address in the following format: `<NEXUS_URL>:<REPOSITORY_PORT>/<PATH>`.

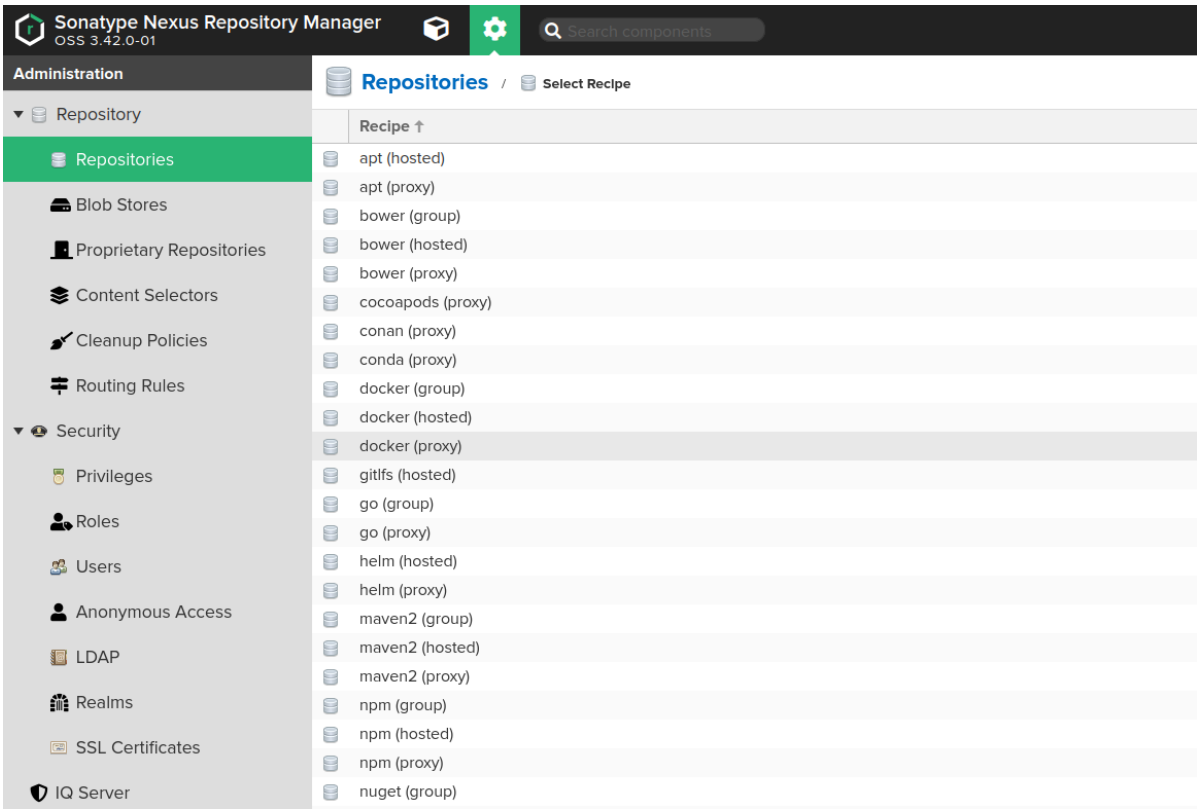
Using the `URL` value from the Nexus repository settings is **not allowed**.

When using the Nexus repository manager, the following requirements must be met:

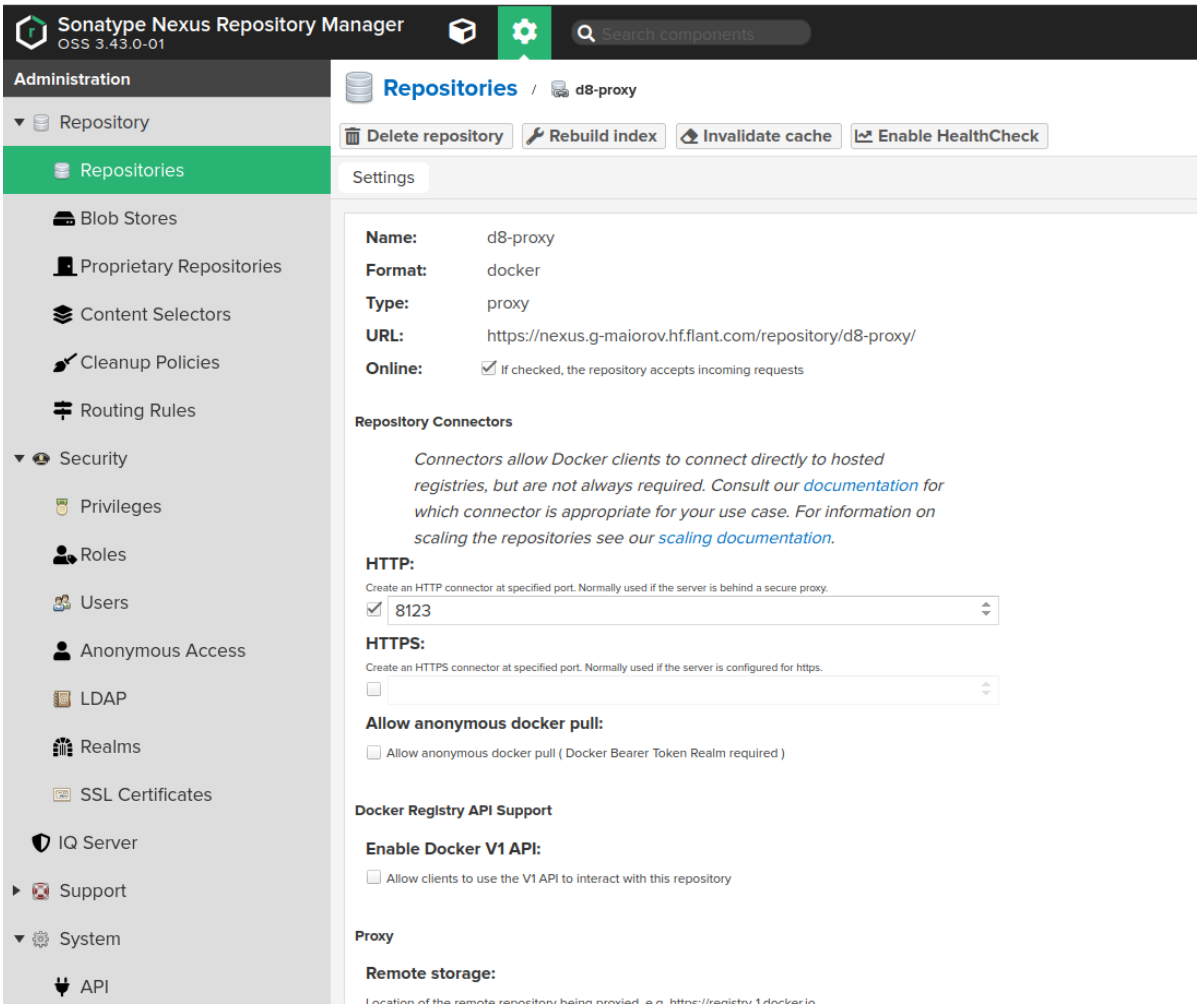
- A proxy Docker repository must be created ("Administration" → "Repository" → "Repositories"):
  - The `Maximum metadata age` parameter is set to `0`.
- Access control must be configured:
  - Create a role named **Nexus** ("Administration" → "Security" → "Roles") with the following privileges:
    - `nx-repository-view-docker-<repository>-browse`
    - `nx-repository-view-docker-<repository>-read`
  - Create a user ("Administration" → "Security" → "Users") and assign them the **Nexus** role.

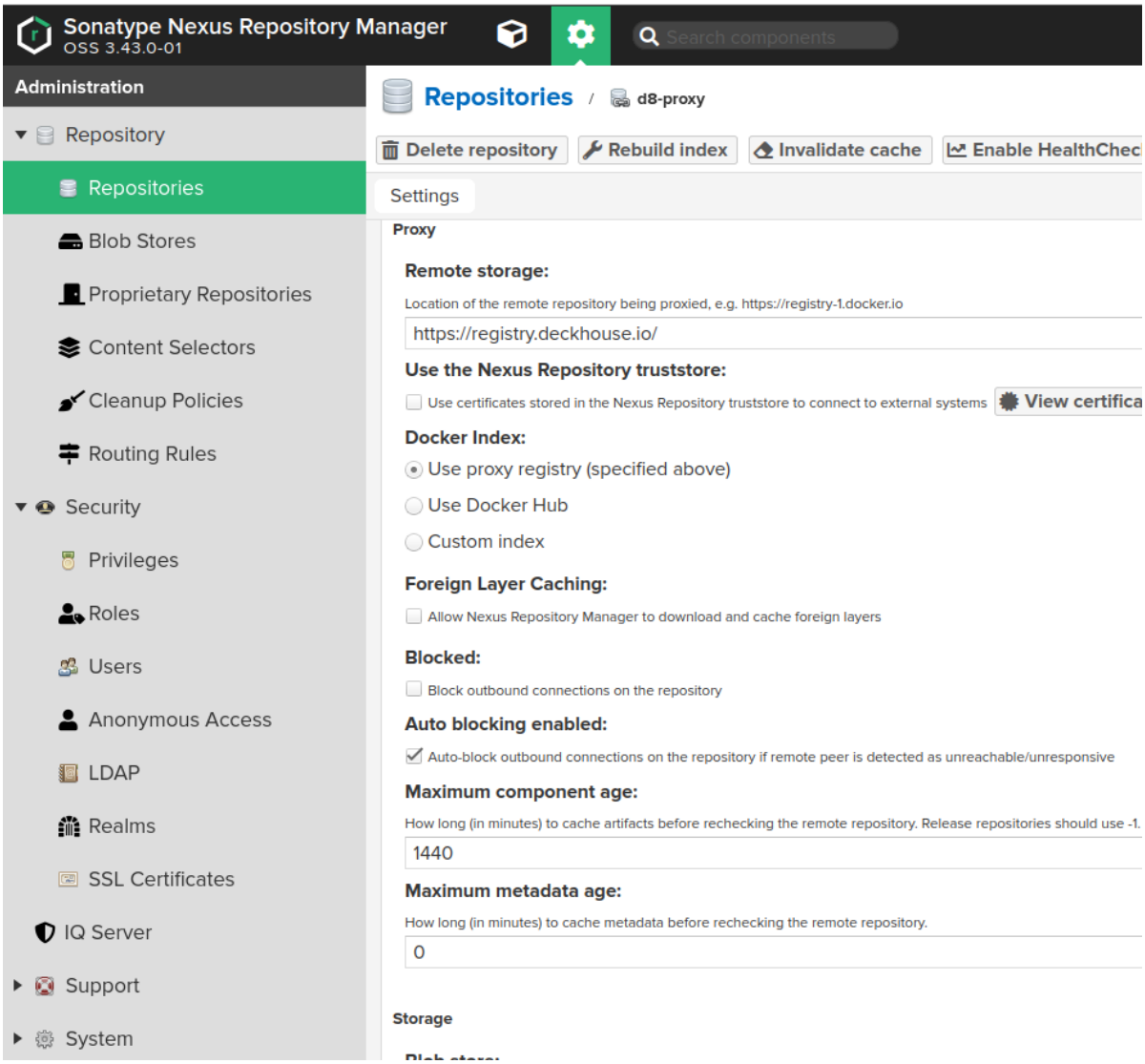
To configure Nexus, follow these steps:

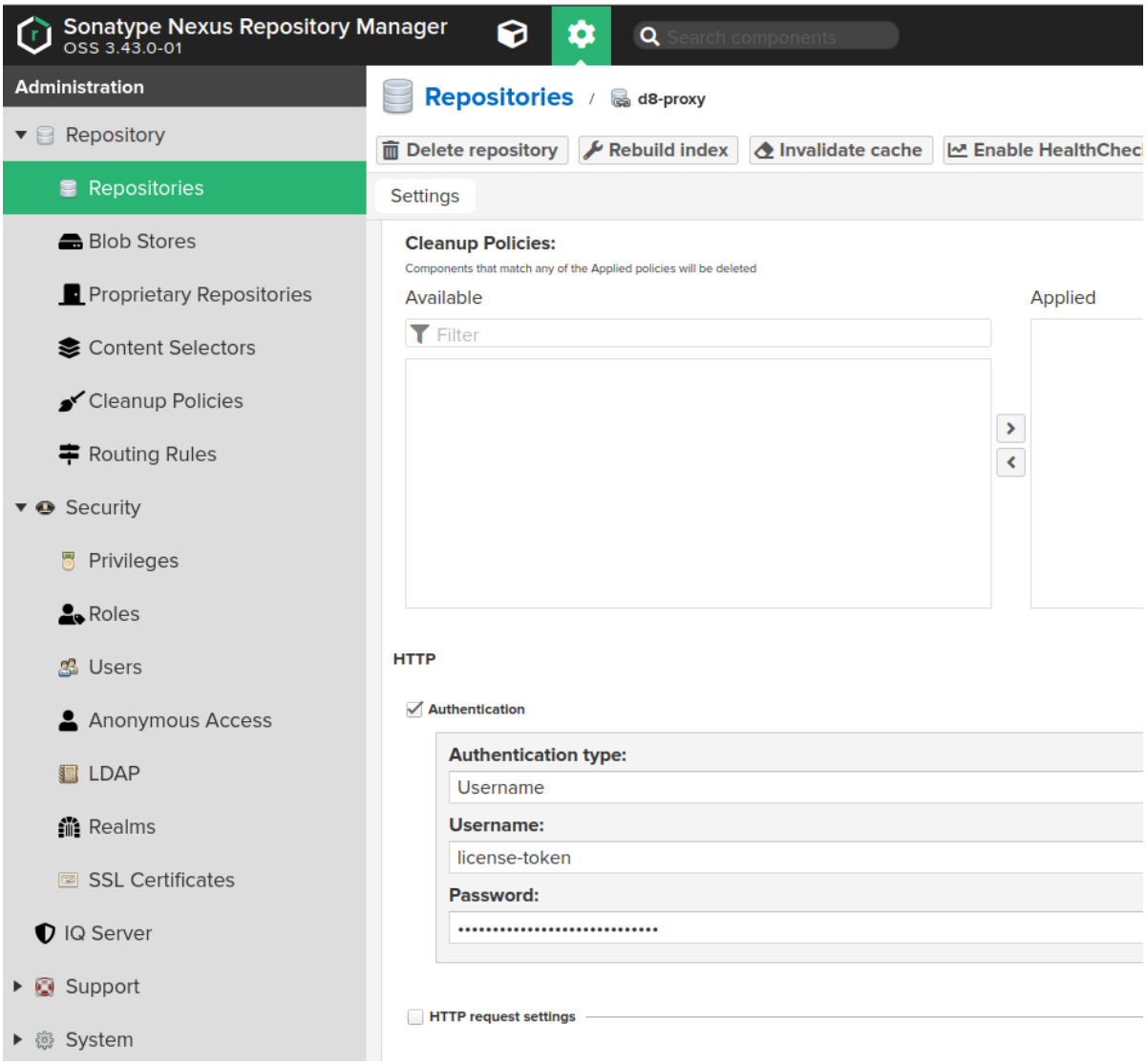
1. Create a proxy Docker repository ("Administration" → "Repository" → "Repositories") that points to the public Deckhouse container registry.



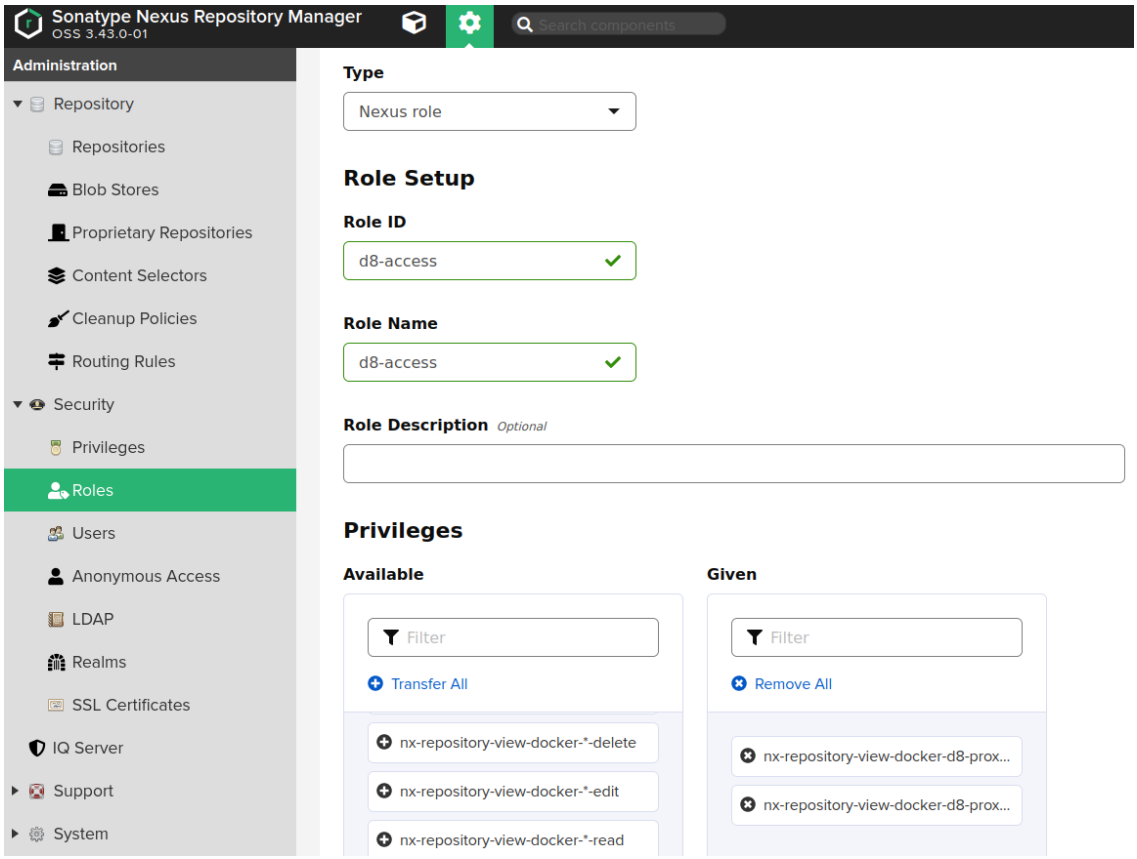
2. Fill out the repository creation form with the following values:
   - `Name`: the desired repository name, e.g., `d8-proxy`.
   - `Repository Connectors / HTTP` or `HTTPS`: a dedicated port for the new repository, e.g., `8123` or another.
   - `Remote storage`: must be set to `https://registry.deckhouse.io/`.
   - `Auto blocking enabled` and `Not found cache enabled`: can be disabled for debugging; otherwise, enable them.
   - `Maximum Metadata Age`: must be set to `0`.
   - If using a commercial edition of Deckhouse Platform, enable the `Authentication` checkbox and fill in the following:
     - `Authentication Type`: `Username`.
     - `Username`: `license-token`.
     - `Password`: Your Deckhouse Platform license key.


Sonatype Nexus Repository Manager
OSS 3.43.0-01
Search components
Administration
Repository
Repositories
Blob Stores
Proprietary Repositories
Content Selectors
Cleanup Policies
Routing Rules
Security
Privileges
Roles
Users
Anonymous Access
LDAP
Realms
SSL Certificates
IQ Server
Support
System
API
Repositories / d8-proxy
Delete repository
Rebuild index
Invalidate cache
Enable HealthCheck
Settings
Name: d8-proxy
Format: docker
Type: proxy
URL: https://nexus.g-maiorov.hf.flant.com/repository/d8-proxy/
Online: If checked, the repository accepts incoming requests
Repository Connectors
Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case. For information on scaling the repositories see our scaling documentation.
HTTP:
Create an HTTP connector at specified port. Normally used if the server is behind a secure proxy.
8123
HTTPS:
Create an HTTPS connector at specified port. Normally used if the server is configured for https.
Allow anonymous docker pull:
Allow anonymous docker pull (Docker Bearer Token Realm required)
Docker Registry API Support
Enable Docker V1 API:
Allow clients to use the V1 API to interact with this repository
Proxy
Remote storage:

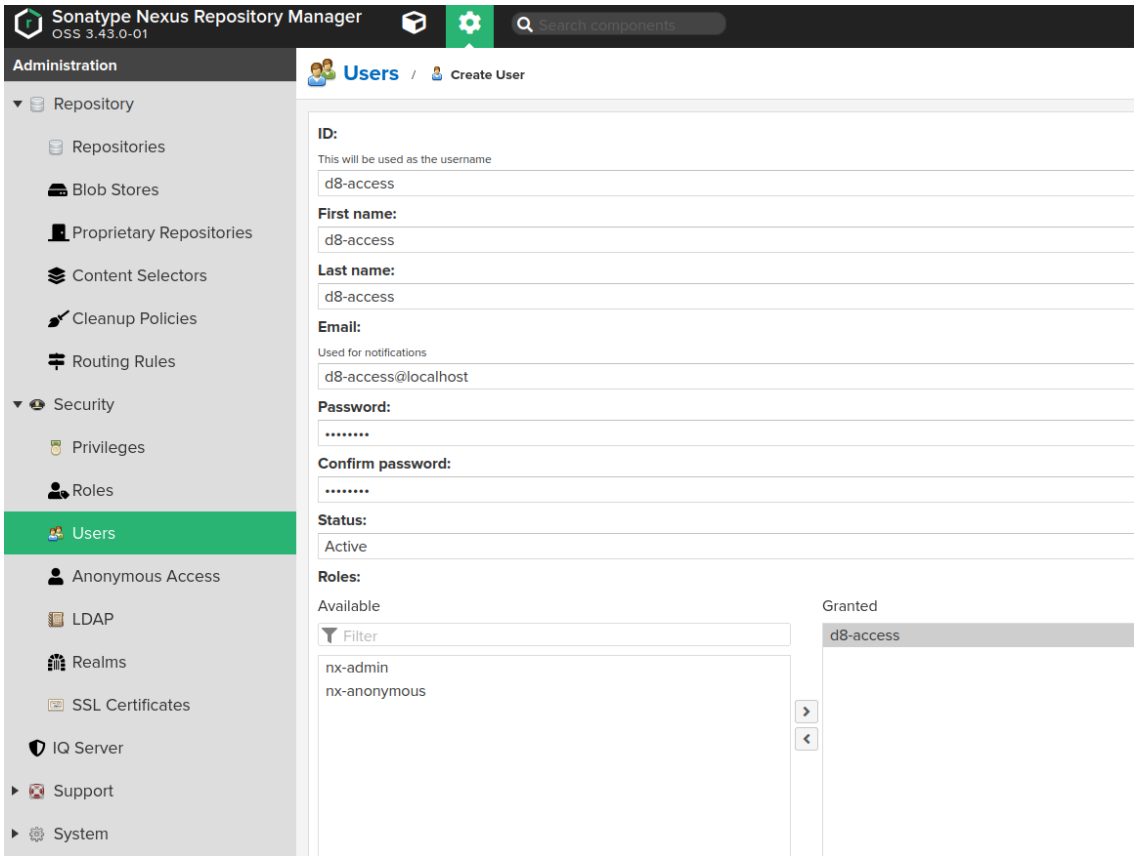

Sonatype Nexus Repository Manager
OSS 3.43.0-01
Search components
Administration
Repository
Repositories
Blob Stores
Proprietary Repositories
Content Selectors
Cleanup Policies
Routing Rules
Security
Privileges
Roles
Users
Anonymous Access
LDAP
Realms
SSL Certificates
IQ Server
Support
System
Repositories / d8-proxy
Delete repository
Rebuild index
Invalidate cache
Enable HealthChec
Settings
Proxy
Remote storage:
Location of the remote repository being proxied, e.g. https://registry-1.docker.io
https://registry.deckhouse.io/
Use the Nexus Repository truststore:
Use certificates stored in the Nexus Repository truststore to connect to external systems
View certifica
Docker Index:
Use proxy registry (specified above)
Use Docker Hub
Custom index
Foreign Layer Caching:
Allow Nexus Repository Manager to download and cache foreign layers
Blocked:
Block outbound connections on the repository
Auto blocking enabled:
Auto-block outbound connections on the repository if remote peer is detected as unreachable/unresponsive
Maximum component age:
How long (in minutes) to cache artifacts before rechecking the remote repository. Release repositories should use -1.
1440
Maximum metadata age:
How long (in minutes) to cache metadata before rechecking the remote repository.
0
Storage



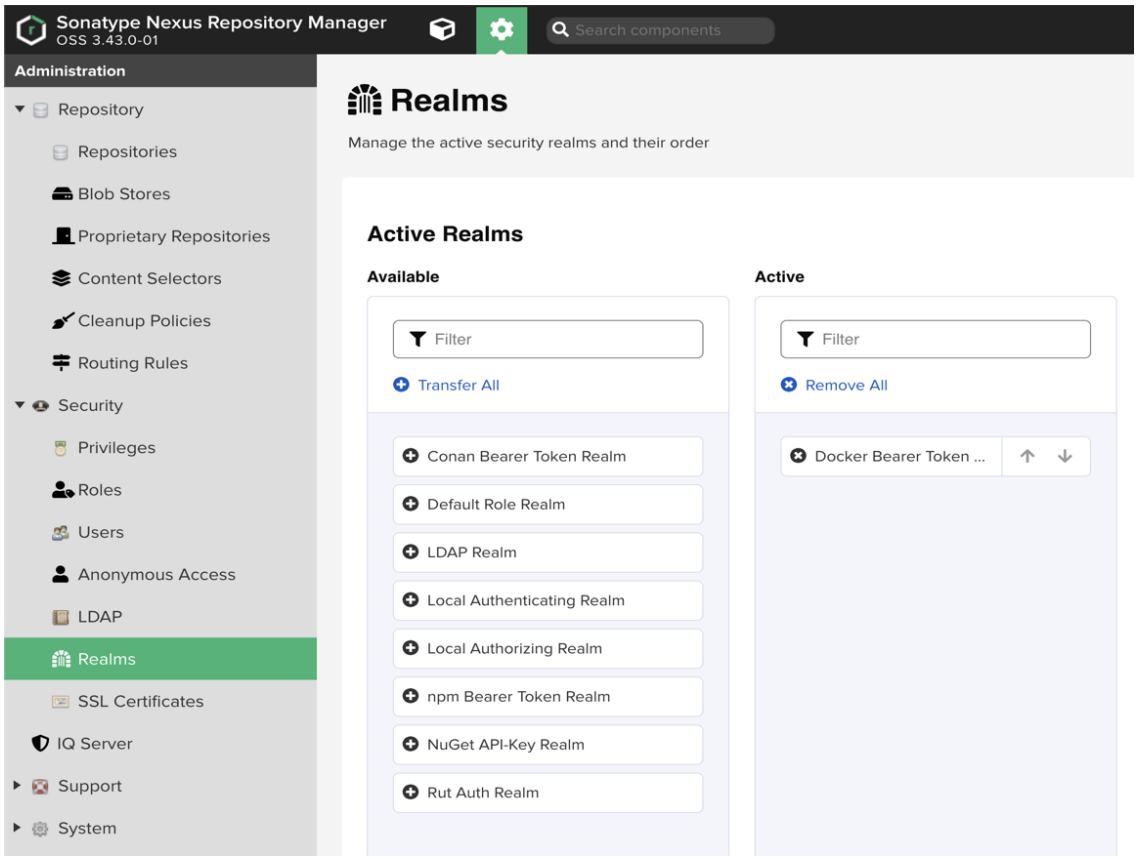
3. Configure Nexus access control to allow Deckhouse Platform to access the created repository:
   - Create a **Nexus** role (“Administration” → “Security” → “Roles”) with the following privileges: `nx-repository-view-docker-<repository>-browse` and `nx-repository-view-docker-<repository>-read`.



- Create a user (“Administration” → “Security” → “Users”) and assign them the role created above.



- Enable **Docker Bearer Token Realm** (“Administration” → “Security” → “Realms”):
  - The **Docker Bearer Token Realm** must be in the **Active** list (on the right), not in **Available** (on the left).
  - If it is not in **Active**:
    1. Find it in the **Available** list.
    2. Move it to **Active** using the arrow button.
    3. Click **Save**.
    4. **Restart Nexus** (it is required for the changes to take effect).

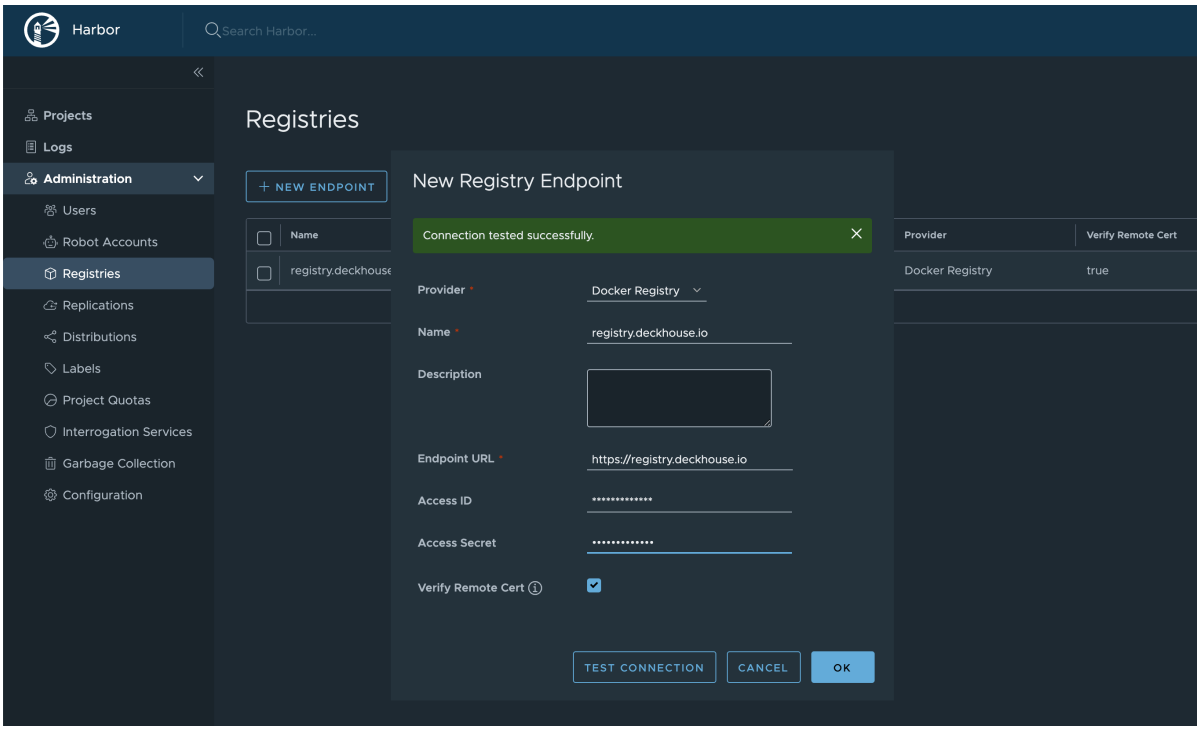


As a result, Deckhouse Platform images will be available at a URL as follows: `https://<NEXUS_HOST>:<REPOSITORY_PORT>/deckhouse/ee:<d8s-version>`.

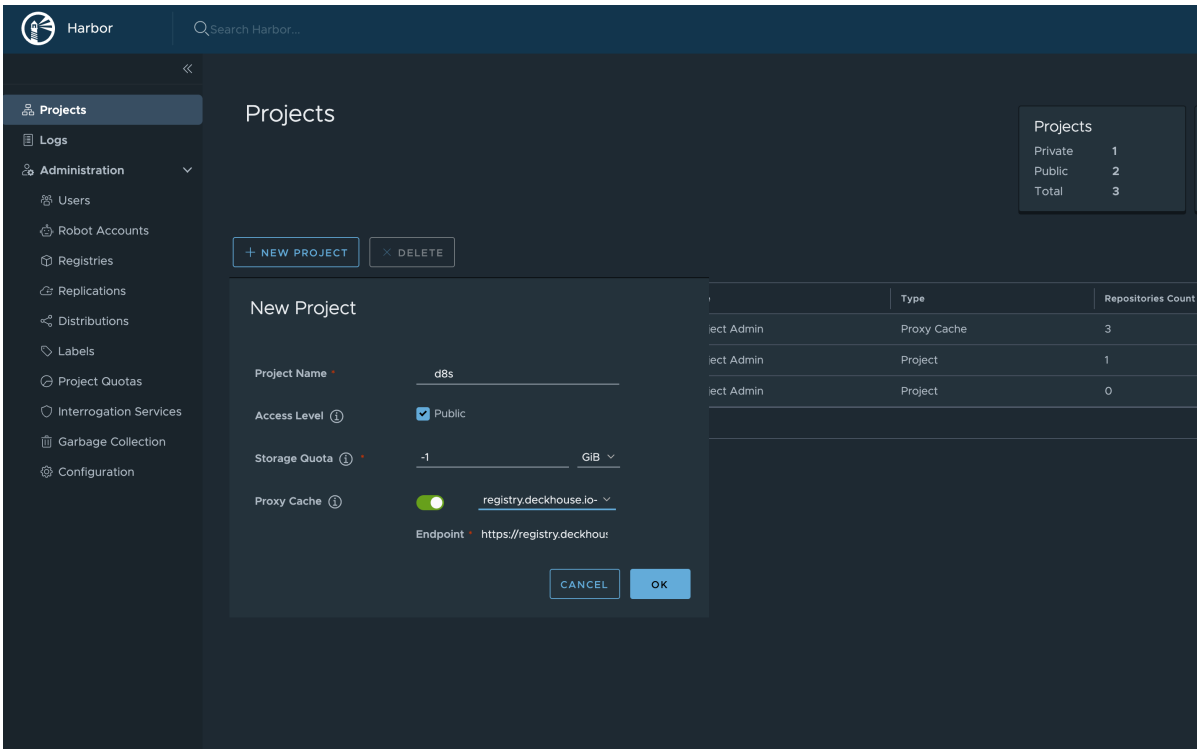
Harbor configuration notes

Use the Harbor Proxy Cache feature.

1. Configure the registry access:
   - In the side menu, navigate to “Administration” → “Registries” and click “New Endpoint” to add a new endpoint for the registry.
   - In the “Provider” dropdown list, select “Docker Registry”.
   - In the “Name” field, enter an endpoint name of your choice.
   - In the “Endpoint URL” field, enter `https://registry.deckhouse.io`.
   - In the “Access ID” field, enter `license-token`.
   - In the “Access Secret” field, enter your Deckhouse Platform license key.
   - Set any remaining parameters as necessary.
   - Click “OK” to confirm creation of a new endpoint for the registry.



2. Create a new project:
   - In the side menu, navigate to “Projects” and click “New Project” to add a project.
   - In the “Project Name” field, enter a project name of your choice (for example, `d8s`). This name will be a part of the URL.
   - In the “Access Level” field, select “Public”.
   - Enable “Proxy Cache” and in the dropdown list, select the registry created earlier.
   - Set any remaining parameters as necessary.
   - Click “OK” to confirm creation of a new project.



Once Harbor is configured, Deckhouse Platform images will be available at a URL as follows: `https://your-harbor.com/d8s/deckhouse/ee:{d8s-version}`.

Manual loading of Deckhouse Platform images and vulnerability DB into a private registry

> **i** You can check the current status of versions in the release channels at releases.deckhouse.io.

- Download and install the Deckhouse CLI utility.
- Download Deckhouse Platform images to a dedicated directory using the `d8 mirror pull` command.

  By default, `d8 mirror pull` downloads only the current versions of Deckhouse Platform, vulnerability scanner databases (if included in the Deckhouse Platform edition), and officially delivered modules. For example, for Deckhouse Platform 1.59, only version 1.59.12 will be downloaded, as it is sufficient for upgrading the platform from 1.58 to 1.59.

  Run the following command (specify the edition code and license key) to download the current version images:

  ```
  d8 mirror pull \
    --source='registry.deckhouse.io/deckhouse/<EDITION>' \
    --license='<LICENSE_KEY>' /home/user/d8-bundle
  ```

  Where:

  - `--source`: Address of the Deckhouse container registry.
  - `<EDITION>`: Deckhouse Platform edition code (for example, `ee`, `se`, `se-plus`). By default, the `--source` parameter refers to the Enterprise Edition (`ee`) and can be omitted.
  - `--license`: Parameter for specifying the Deckhouse Platform license key for authentication in the official container registry.
  - `<LICENSE_KEY>`: Deckhouse Platform license key.
  - `/home/user/d8-bundle`: Directory where the image packages will be placed. It will be created if it does not exist.

  > **i** If the image download is interrupted, rerunning the command will resume the download, provided no more than one day has passed since the interruption.

  Example command to download all Deckhouse Platform EE versions starting from version 1.59 (specify your license key):

  ```
  d8 mirror pull \
  --license='<LICENSE_KEY>' \
  --since-version=1.59 /home/user/d8-bundle
  ```

  Example command to download the current Deckhouse Platform SE versions (specify your license key):

  ```
  d8 mirror pull \
  --license='<LICENSE_KEY>' \
  --source='registry.deckhouse.io/deckhouse/se' \
  /home/user/d8-bundle
  ```

  Example command to download Deckhouse Platform images from an external registry:

  ```
  d8 mirror pull \
  --source='corp.company.com:5000/sys/deckhouse' \
  --source-login='<USER>' --source-password='<PASSWORD>' /home/user/d8-bundle
  ```

  Example command to download the vulnerability scanner database package:

  ```
  d8 mirror pull \
  --license='<LICENSE_KEY>' \
  --no-platform --no-modules /home/user/d8-bundle
  ```

  Example command to download all available additional module packages:

  ```
  d8 mirror pull \
  --license='<LICENSE_KEY>' \
  --no-platform --no-security-db /home/user/d8-bundle
  ```

Example command to download module packages `stronghold` and `secrets-store-integration`:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold \
--include-module secrets-store-integration \
/home/user/d8-bundle
```

Example command to download `stronghold` module with semver `^` constraint (allows updates that don't change the leftmost non-zero digit) from version 1.2.0:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold@1.2.0 \
/home/user/d8-bundle
```

Example command to download `secrets-store-integration` module with semver `~` constraint (allows updates that don't change the last specified digit's place) from version 1.1.0:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module secrets-store-integration@~1.1.0 \
/home/user/d8-bundle
```

Quote the `--include-module` flag value when it contains `>=` or `<=`. Unquoted, the shell treats it as an input/output redirection.

Example command to download the `console` module with semver `>=` constraint (any version from the specified one and above) from version 1.43.2:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module "console@>=1.43.2" \
/home/user/d8-bundle
```

Example command to download exact version of `stronghold` module 1.2.5 and publish to all release channels:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold@=v1.2.5 \
/home/user/d8-bundle
```

Other command parameters available for use:

- `--no-pull-resume`: Force the download to start from the beginning.
- `--force`: Overwrite existing packages if they conflict with the current pull operation.
- `--ignore-suspend`: Ignore suspended release channels and continue mirroring. Use with caution.
- `--no-platform`: Skip downloading the Deckhouse Platform image package (`platform.tar`).
- `--no-modules`: Skip downloading module packages (`module-*.tar`).
- `--no-security-db`: Skip downloading the vulnerability scanner database package (`security.tar`).
- `--no-packages`: Skip downloading Deckhouse packages.
- `--no-installer`: Skip downloading Deckhouse installer images.
- `--only-extra-images`: Pull only extra module images without pulling main module images.
- `--skip-vex-images`: Skip downloading VEX images.
- `--include-platform = CONSTRAINT`: Download Deckhouse Platform releases by a semver constraint. Cannot be used together with `--since-version` and `--deckhouse-tag`. Always quote the constraint value: `>` and `<` are shell redirections. Examples: `--include-platform ">=1.64 <=1.68"`, `--include-platform "~1.65.0"`, `--include-platform "^1.65.0"`, `--include-`

`platform "1.65.0"`, `--include-platform "=v1.65.3"`, or `--include-platform "=v1.65.3+stable"`.

- `--include-module` / `-i = name[@Major.Minor]`: Download only a specific set of modules using a whitelist (and, if needed, their minimum versions). Use multiple times to add more modules to the whitelist. These flags are ignored if used with `--no-modules`.

  The following syntax options are supported for specifying module versions. When using `>=` or `<=`, quote the whole flag value:

  - `module-name@1.3.0`: Pulls versions with semver ^ constraint (^1.3.0), including v1.3.0, v1.3.3, v1.4.1.
  - `module-name@~1.3.0`: Pulls versions with semver ~ constraint (>=1.3.0 <1.4.0), including only v1.3.0, v1.3.3.
  - `"module-name@>=1.3.0"`: Pulls versions with semver `>=` constraint, including the explicitly specified version and newer versions matching the constraint.
  - `"module-name@>=1.3.0 <=1.4.0"`: Pulls versions in the range, honoring both the lower and upper boundaries.
  - `module-name@=v1.3.0`: Pulls exact tag match v1.3.0, publishing to all release channels.
  - `module-name@=v1.3.0+stable`: Pulls exact tag match v1.3.0, publishing to the stable release channel.
  - `module-name@=bobV1`: Pulls exact tag match `bobV1`, publishing to all release channels.
- `--exclude-module` / `-e = name`: Skip downloading a specific set of modules using a blacklist. Use multiple times to add more modules to the blacklist. Ignored if `--no-modules` or `--include-module` is used;
- `--include-package = name[@version]`: Download only a specific set of packages using a whitelist. Versions and semver constraints use the same syntax as `--include-module`, including quoting rules.
- `--exclude-package = name[@version]`: Skip downloading a specific set of packages using a blacklist. Ignored if `--include-package` is used.
- `--modules-path-suffix`: Change the suffix of the path to the module repository in the main Deckhouse Platform registry. The default suffix is `/modules` (for example, the full path to the module repo will be `registry.deckhouse.io/deckhouse/EDITION/modules`).
- `--since-version=X.Y`: Download all Deckhouse Platform versions starting from the specified minor version. This option is ignored if the specified version is higher than the version on the Rock Solid release channel. Cannot be used with `--deckhouse-tag`.
- `--deckhouse-tag`: Download only the specific Deckhouse Platform version (regardless of release channels). Cannot be used with `--since-version`.
- `--installer-tag=TAG`: Download a specific Deckhouse installer tag. Defaults to `latest` if omitted.
- `--proxy-registry`: Pull from a proxy/cache registry that does not support the registry catalog API. Requires `--include-platform` unless the platform is skipped with `--no-platform`, and at least one `--include-module` unless modules are skipped with `--no-modules`. Cannot be used together with `--deckhouse-tag` or `--since-version`.
- `--dry-run`: Print what would be pulled without downloading images.
- `--verbose-summary`: Print a detailed summary for every module and package with resolved versions.
- `--gost-digest`: Calculate the checksum of the final Deckhouse Platform image bundle using the GOST R 34.11-2012 (Streebog) algorithm. The checksum will be displayed and written to a `.tar.gostsum` file in the folder containing the image tarball.
- `--source-login` and `--source-password`: Authentication data to access the external registry.
- `--tls-skip-verify`: Disable TLS certificate validation.
- `--insecure`: Interact with registries over HTTP.
- `--images-bundle-chunk-size=N`: The maximum file size (in GB) to split the image archive into. As a result, instead of one image archive, a set of CHUNK files will be created (for example, `d8.tar.NNNN.chunk`). To upload images from such a set, use the file name without the `.NNNN.chunk` suffix (for example, `d8.tar` for files `d8.tar.NNNN.chunk`).
- `--tmp-dir`: Path to a directory for temporary files used during image download and upload. All processing is done in this directory. It must have enough free disk space to hold the entire image bundle. Defaults to the `.tmp` subdirectory in the image bundle directory.

Additional configuration parameters for the `d8 mirror` command family are available as environment variables:

- `HTTP_PROXY` / `HTTPS_PROXY`: Proxy server URL for HTTP(S) requests not listed in the `$NO_PROXY` variable.
- `NO_PROXY`: Comma-separated list of hosts to exclude from proxying. Each entry can be an IP (`1.2.3.4`), CIDR (`1.2.3.4/8`), domain, or wildcard (`*`). IPs and domains may include a port (`1.2.3.4:80`). A domain matches itself and all subdomains. A domain starting with a `.` matches only subdomains. For example, `foo.com` matches `foo.com` and `bar.foo.com`; `.y.com` matches `x.y.com` but not `y.com`. The `*` disables proxying.
- `SSL_CERT_FILE`: Path to an SSL certificate. If set, system certificates are not used.
- `SSL_CERT_DIR`: Colon-separated list of directories to search for SSL certificate files. If set, system

certificates are not used. More info...
- `MIRROR_BYPASS_ACCESS_CHECKS`: Set this variable to `1` to disable credential validation for the registry.
- On the host with access to the container registry where Deckhouse Platform images should be uploaded, copy the downloaded Deckhouse Platform image bundle and install the Deckhouse CLI.
- Upload Deckhouse Platform images to the registry using the `d8 mirror push` command.

  The `d8 mirror push` command uploads images from all packages located in the specified directory. If you only want to push specific packages, you can either run the command separately for each TAR image bundle by specifying the direct path to it, or temporarily remove the `.tar` extension from unwanted files or move them out of the directory.

  Example command to upload image packages from the `/mnt/MEDIA/d8-images` directory (provide authentication data if required):

  ```
  d8 mirror push /mnt/MEDIA/d8-images 'corp.company.com:5000/sys/deckhouse' \
    --registry-login='<USER>' --registry-password='<PASSWORD>'
  ```

  Before uploading the images, make sure that the target path in the container registry exists (in the example, `/sys/deckhouse`) and that the account used has write permissions.

  If you're using Harbor, you won't be able to upload images to the root of a project. Use a dedicated repository within the project to store Deckhouse Platform images.
- After uploading the images to the registry, you can proceed with installing Deckhouse Platform. Use the getting-started guide.

  When running the installer, use the address of your own image registry (where the images were uploaded earlier) instead of the official public Deckhouse Platform container registry. For the example above, the installer image address will be `corp.company.com:5000/sys/deckhouse/install:stable` instead of `registry.deckhouse.io/deckhouse/ee/install:stable`.

  In the `registry` section of ModuleConfig `deckhouse`, use your registry address and authorization data (starting with Deckhouse Platform 1.75). The legacy method is to use InitConfiguration (parameters `imagesRepo`, `registryDockerCfg`).

Creating a cluster and running Deckhouse Platform without using release channels

⚠ This method should only be used if your private (isolated) registry does not contain images with release channel metadata.

If you need to install Deckhouse Platform with automatic updates disabled:

1. Use the installer image tag corresponding to the desired version. For example, to install release `v1.44.3`, use the image `your.private.registry.com/deckhouse/install:v1.44.3`.
2. Specify the appropriate version number in the `deckhouse.devBranch` parameter of InitConfiguration.

   ⚠ **Do not specify** the `deckhouse.releaseChannel` parameter in InitConfiguration.

If you want to disable automatic updates in an already running Deckhouse Platform installation (including patch updates), remove the `releaseChannel` parameter from the `deckhouse` module configuration.

Using a proxy server

Example steps for configuring a proxy server using Squid...

1. Prepare a server (or virtual machine). The server must be reachable from the required cluster nodes and must have internet access.
2. Install Squid (the following examples are for Ubuntu):

```
apt-get install squid
```

3. Create the Squid configuration file:

```
cat <<EOF > /etc/squid/squid.conf
auth_param basic program /usr/lib/squid3/basic_ncsa_auth /etc/squid/passwords
auth_param basic realm proxy
acl authenticated proxy_auth REQUIRED
http_access allow authenticated
# Specify the required port. Port 3128 is used by default.
http_port 3128
```

4. Create a username and password for proxy server authentication:

   Example for user `test` with password `test` (make sure to change it):

```
echo "test:$(openssl passwd -crypt test)" >> /etc/squid/passwords
```

5. Start Squid and enable it to start on server boot:

```
systemctl restart squid
systemctl enable squid
```

To configure Deckhouse Platform to work with a proxy server, use the `proxy` parameter of the ClusterConfiguration resource.

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: OpenStack
  prefix: main
cri: "Containerd"
proxy:
  httpProxy: "http://user:password@proxy.company.my:3128"
  httpsProxy: "https://user:password@proxy.company.my:8443"
---
# control-plane-manager module settings.
# https://deckhouse.io/modules/control-plane-manager/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "Default"
    network:
      podSubnetCIDR: 10.111.0.0/16
      serviceSubnetCIDR: 10.222.0.0/16
      clusterDomain: "cluster.local"
```

Automatic proxy variable loading for users in CLI

Starting from version 1.67, the `/etc/profile.d/d8-system-proxy.sh` file is no longer configured in Deckhouse Platform to set proxy variables for users.
To automatically load proxy variables for users in CLI, use the NodeGroupConfiguration resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: profile-proxy.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 99
  content: |
    {{- if .proxy }}
      {{- if .proxy.httpProxy }}
    export HTTP_PROXY={{ .proxy.httpProxy | quote }}
    export http_proxy=${HTTP_PROXY}
      {{- end }}
      {{- if .proxy.httpsProxy }}
    export HTTPS_PROXY={{ .proxy.httpsProxy | quote }}
    export https_proxy=${HTTPS_PROXY}
      {{- end }}
      {{- if .proxy.noProxy }}
    export NO_PROXY={{ .proxy.noProxy | join "," | quote }}
    export no_proxy=${NO_PROXY}
      {{- end }}
    bb-sync-file /etc/profile.d/profile-proxy.sh - << EOF
    export HTTP_PROXY=${HTTP_PROXY}
    export http_proxy=${HTTP_PROXY}
    export HTTPS_PROXY=${HTTPS_PROXY}
    export https_proxy=${HTTPS_PROXY}
    export NO_PROXY=${NO_PROXY}
    export no_proxy=${NO_PROXY}
    EOF
    {{- else }}
    rm -rf /etc/profile.d/profile-proxy.sh
    {{- end }}
```

Additional resources

- Getting started
- Supported Kubernetes and OS versions
- Integration with IaaS providers
- Installing Deckhouse Platform in a private environment
- Going to Production
- Global custom resources
- Global parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

How to configure?

Deckhouse configuration

You can configure Deckhouse using:

- **Global settings**. Global settings are stored in the `ModuleConfig/global` custom resource. Global settings can be be thought of as a special `global` module that cannot be disabled.
- **Module settings**. Module settings are stored in the `ModuleConfig` custom resource; its name is

the same as that of the module (in kebab-case).
- **Custom resources.** Some modules are configured using the additional custom resources.

An example of a set of custom resources for configuring Deckhouse:

```
# Global settings.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 1
  settings:
    modules:
      publicDomainTemplate: "%s.kube.company.my"
---
# The monitoring-ping module settings.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  settings:
    externalTargets:
    - host: 8.8.8.8
---
# Disable the dashboard module.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: dashboard
spec:
  enabled: false
```

You can view the list of `ModuleConfig` custom resources and the states of the corresponding modules (enabled/disabled) as well as their statuses using the following command:

```
d8 k get moduleconfigs
```

Example output...

```
$ d8 k get moduleconfigs
NAME             ENABLED   VERSION   AGE     MESSAGE
deckhouse        true      1         12h
documentation    true      1         12h
global                     1         12h
prometheus       true      2         12h
upmeter          false     2         12h
```

To change the global Deckhouse configuration or module configuration, create or edit the corresponding `ModuleConfig` custom resource.

For example, this command allows you to configure the `upmeter` module:

```
d8 k edit moduleconfig/upmeter
```

Changes are applied automatically once the resource configuration is saved.

Modifying cluster configuration

⚠ To apply changes related to node configuration, you must run the `dhctl converge` command using the DP installer.
This command synchronizes the actual node state with the specified configuration.

General cluster parameters are defined in the ClusterConfiguration structure.

To modify these parameters, run the following command:

```
d8 system edit cluster-configuration
```

After saving the changes, DP will automatically reconcile the cluster state with the new configuration. Depending on the cluster size, this process may take some time.

Modifying protected parameters

Some cluster parameters are critical for cluster operation and cannot be changed in a running cluster by default. These parameters include:

- `podSubnetCIDR`: The Pod network address space.
- `podSubnetNodeCIDRPrefix`: The Pod network prefix size per node.
- `serviceSubnetCIDR`: the Service network address space.

Attempts to change these parameters will be blocked by the admission webhook with an error message.

✖ **Changing these parameters in a running cluster can lead to:**

- Complete loss of access to the Kubernetes API.
- Invalidation of TLS certificates.
- Necessity to restart all cluster nodes and control plane components.
- Data inconsistency if the process is interrupted.

It is recommended to recreate the cluster instead of changing these parameters.

If you absolutely must change these parameters (e.g., for testing or in exceptional circumstances), you can bypass the protection mechanism.

⚠ Even with the protection mechanism bypassed, there is **no guarantee** that the cluster will continue to function correctly after changing these parameters. Be prepared for the possibility of complete cluster failure and have a backup plan.

Modifying protected parameters using dhctl

Use the `dhctl` tool from the DP installer container with the `--yes-i-am-sane-and-i-understand-what-i-am-doing` flag.

It will automatically:

- Add the `deckhouse.io/allow-unsafe` annotation to the `d8-cluster-configuration` Secret;
- Open an editor to modify the configuration;
- Remove the annotation after you save the changes.

To change protected settings using `dhctl`, follow these steps:

1. Get the current DP version and edition from your cluster. To do this, run the DP installer container of the appropriate edition and version **on your local machine**:

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]')
   ```

2. Run the DP installer container (adjust the registry address if needed):

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

Where `<MOUNT_OPTIONS>` — mounting parameters for files in the installer container, such as:

- SSH access keys;
- Configuration file;
- Resources file, etc.

3. Inside the container, run the following command to edit the cluster configuration:

```
dhctl config edit cluster-configuration \
  --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
  --ssh-user=<USERNAME> \
  --ssh-host=<MASTER-NODE-HOST> \
  --yes-i-am-sane-and-i-understand-what-i-am-doing
```

Where:

- `<SSH_KEY_FILENAME>`: Your SSH private key filename;
- `<USERNAME>`: SSH user with sudo privileges on the target master node of the cluster;
- `<MASTER-NODE-HOST>` — master node IP address or hostname.

4. Edit the configuration in the opened editor, save, and exit the editor.

Manually changing protected parameters

Manually edit the configuration:

1. Add the `deckhouse.io/allow-unsafe` annotation to the `d8-cluster-configuration` Secret:

```
d8 k -n kube-system annotate secret d8-cluster-configuration deckhouse.io/allow-unsafe="true"
```

2. Get the current configuration, decode it, and save to a file:

```
d8 k -n kube-system get secret d8-cluster-configuration \
  -o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d > cluster-config.yaml
```

3. Edit the `cluster-config.yaml` file with your preferred editor:

```
vi cluster-config.yaml
```

4. Encode the edited configuration and update the Secret:

```
d8 k -n kube-system patch secret d8-cluster-configuration \
  --patch="{\"data\":{\"cluster-configuration.yaml\":\"$(base64 -w0 < cluster-
config.yaml)\"}}"
```

5. Remove the annotation after applying the changes:

```
d8 k -n kube-system annotate secret d8-cluster-configuration deckhouse.io/allow-unsafe-
```

6. Delete the temporary file:

```
rm cluster-config.yaml
```

⚠ If you forget to remove the `deckhouse.io/allow-unsafe` annotation, this protection mechanism will remain disabled, leaving your cluster vulnerable to accidental configuration changes.

Viewing current configuration

DP is managed through global settings, module configurations, and various custom resources.

1. To view global settings, run:

   ```
   d8 k get mc global -o yaml
   ```

2. To view the status of all modules (available in Deckhouse version 1.47+):

   ```
   d8 k get modules
   ```

3. To view the configuration of the `user-authn` module:

   ```
   d8 k get moduleconfigs user-authn -o yaml
   ```

Configuring the module

The module is configured using the ModuleConfig custom resource , whose name is the same as the module name (in kebab-case). The ModuleConfig custom resource has the following fields:

- `metadata.name` — the name of the module in kebab-case (e.g, `prometheus`, `node-manager`).
- `spec.version` — version of the module settings schema. It is an integer greater than zero. This field is mandatory if `spec.settings` is not empty. You can find the latest version number in the module's documentation under *Settings*.
  - Deckhouse is backward-compatible with older versions of the module's settings schema. If an outdated version of the schema is used, a warning stating that you need to update the module's schema will be displayed when editing or viewing the custom resource.
- `spec.settings` — module settings. This field is optional if the `spec.enabled` field is used. For a description of the available settings, see *Settings* in the module's documentation.
- `spec.enabled` — this optional field allows you to explicitly enable or disable the module. The module may be enabled by default based on the bundle in use if this parameter is not set.

> **i** Deckhouse doesn't modify ModuleConfig resources. As part of the Infrastructure as Code (IaC) approach, you can store ModuleConfigs in a version control system and use Helm, `d8 k`, and other familiar tools for deploy.

An example of a custom resource for configuring the `kube-dns` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  settings:
    stubZones:
    - upstreamNameservers:
      - 192.168.121.55
      - 10.2.7.80
      zone: directory.company.my
    upstreamNameservers:
    - 10.2.100.55
    - 10.2.200.55
```

Some modules can also be configured using custom resources. Use the search bar at the top of the page or select a module in the left menu to see a detailed description of its settings and the custom resources used.

Enabling and disabling the module

> **i** Depending on the bundle used, some modules may be enabled by default.

To enable/disable the module, set `spec.enabled` field of the ModuleConfig custom resource to `true` or

`false`. Note that this may require you to first create a ModuleConfig resource for the module.

Here is an example of disabling the `user-authn` module (the module will be turned off even if it is enabled as part of a module bundle):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  enabled: false
```

To check the status of the module, run the `d8 k get moduleconfig <MODULE_NAME>` command:

Example:

```
$ d8 k get moduleconfig user-authn
NAME         ENABLED   VERSION   AGE   MESSAGE
user-authn   false     1         12h
```

Module bundles

Depending on the bundle used, modules may be enabled or disabled by default.

⚠ The table below describes module bundles only for built-in modules of Deckhouse Platform. Modules from sources are not included in this table.

Bundle name	List of modules, enabled by default
Default	• node-local-dns • admission-policy-engine • cert-manager • chrony • control-plane-manager • dashboard • deckhouse • documentation • deckhouse-tools • descheduler • extended-monitoring • kube-dns • kube-proxy • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes-control-plane • monitoring-kubernetes • monitoring-ping • multitenancy-manager • namespace-configurator • registry-packages-proxy • node-manager • operator-prometheus • prometheus • prometheus-metrics-adapter • secret-copier • terraform-manager • user-authn • user-authz • vertical-pod-autoscaler • registry

Bundle name	List of modules, enabled by default
Managed	• admission-policy-engine • cert-manager • dashboard • deckhouse • documentation • deckhouse-tools • descheduler • extended-monitoring • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-ping • multitenancy-manager • namespace-configurator • operator-prometheus • prometheus • prometheus-metrics-adapter • secret-copier • user-authz • vertical-pod-autoscaler
Minimal	• deckhouse

Things to keep in mind when working with the Minimal module set

⚠ **Note** that several basic modules are not included in the `Minimal` set of modules (for example, the CNI module).

Deckhouse Platform with the `Minimal` module set and no basic modules included will only be able to operate in an already deployed cluster.

To install DP with the `Minimal` module set, enable at least the following modules by specifying them in the installer configuration file:

- cloud provider module (for example, `cloud-provider-aws` for AWS), in a case of deploying a cloud cluster;
- `cni-cilium` or another CNI control module (if necessary);
- `control-plane-manager`;
- `kube-dns`;
- `node-manager`;
- `registry-packages-proxy`;
- `terraform-manager`, in a case of deploying a cloud cluster.

Accessing documentation for the current version

The documentation for the running version of Deckhouse is available at `documentation.<cluster domain>`,
where `<cluster domain>` is the DNS name generated according to the template specified in the `modules.publicDomainTemplate` parameter of the global configuration.

⚠ Documentation is available only if the `documentation` module is enabled in the cluster.
It is enabled by default, except when using the `Minimal` delivery bundle.

Managing placement of Deckhouse components

Advanced scheduling

If no `nodeSelector/tolerations` are explicitly specified in the module parameters, the following strategy is used for all modules:

1. If the `nodeSelector` module parameter is not set, then Deckhouse will try to calculate the

`nodeSelector` automatically. Deckhouse looks for nodes with the specific labels in the cluster (see the list below). If there are any, then the corresponding `nodeSelectors` are automatically applied to module resources.
2. If the `tolerations` parameter is not set for the module, all the possible tolerations are automatically applied to the module's Pods (see the list below).
3. You can set both parameters to `false` to disable their automatic calculation.
4. If there are no nodes with a specific role in the cluster and `nodeSelector` is automatically selected (see point 1), `nodeSelector` will not be specified in the module resources. The module will then use any node with non-conflicting `taints`.

You cannot set `nodeSelector` and `tolerations` for modules:

- running on all cluster nodes (such as `cni-flannel` or `monitoring-ping`);
- running on all master nodes (such as `prometheus-metrics-adapter` or `vertical-pod-autoscaler`).

Module features that depend on its type

> **i** Below is the basic (general) logic for automatically selecting nodes to place module components when no explicit `nodeSelector` and `tolerations` values are set in the module settings. Some modules may extend or override this logic (for example, by using Kubernetes mechanisms such as affinity/anti-affinity, `topologySpreadConstraints`, or their own node selection rules). See the relevant module documentation for details.

If `nodeSelector` is not explicitly set in the module configuration, Deckhouse determines the placement of components automatically. The order in which nodes are selected depends on the module type.

- The `console` module:
  - If `nodeSelector` is not explicitly set in the module configuration, module components are placed on control plane nodes.
  - If control plane nodes are not available for such placement, nodes with the `node-role.deckhouse.io/system` label are used.
- The *monitoring*-related modules (`operator-prometheus`, `prometheus` and `vertical-pod-autoscaler`):
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE_NAME` label is present in the cluster.
    2. It checks if a node with the `node-role.deckhouse.io/monitoring` label is present in the cluster.
    3. It checks if a node with the `node-role.deckhouse.io/system` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE_NAME"}` (e.g., `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"operator-prometheus"}`).
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"monitoring"}`.
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"system"}`.
- The *frontend*-related modules (`ingress-nginx` only):
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE_NAME` label is present in the cluster.
    2. It checks if a node with the `node-role.deckhouse.io/frontend` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE NAME"}`.
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"frontend"}`.
- Other modules:
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE_NAME` label is present in the cluster (e.g., `node-role.deckhouse.io/cert-manager`).
    2. It checks if a node with the `node-role.deckhouse.io/system` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE_NAME"}` (e.g., `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"network-gateway"}`).
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"system"}`.

Additional resources

- Global parameters
- Global custom resources
- Global custom resources
- Module `kube-dns` documentation
- Module `upmeter` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

DP components registry

This section describes the settings for interacting with the DP component registry.

This section covers configuring DP interaction with the registry in a running cluster. If you need information about working with the registry during cluster installation, go to the “Platform installation” section.

The capabilities and processes for configuring the DP component registry depend on how the cluster is managed. In clusters fully managed by DP, configuration management is handled by the `registry` module (for more details, see the section “Managing the registry in DP-managed clusters”). In Managed Kubernetes clusters, the `helper change-registry` is used; the `registry` module is not used (for more details, see the section “Switching a Managed Kubernetes cluster to use third-party registry”).

Additional resources

- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing the registry in Deckhouse Platform-managed clusters

In clusters fully managed by Deckhouse Platform (Deckhouse Platform), you can configure the download paths:

- images of Deckhouse Platform component containers;
- images of containers from additional repositories (vendor repositories or your own).

The pull paths for container images are managed using the `registry` module.

Implementations of container image pull path management

Deckhouse Platform supports two implementations of the `registry` module for managing container image download paths:

- **The current implementation**. Used starting with Deckhouse Platform 1.78. Container image pull paths are configured through the `registry` ModuleConfig. The implementation is described in the “Current implementation” section.
- **The v1 implementation**. Used in Deckhouse Platform up to and including 1.77. The Deckhouse Platform component registry is configured through the `deckhouse` ModuleConfig. The implementation is described in the “V1 implementation” section.

The two implementations never manage a cluster at the same time: only one of them is active at any given moment. A cluster that has never run the v1 implementation uses the current one right away. A cluster running the v1 implementation keeps using it until the conditions for switching to the current one are met — control is then handed over to the current implementation automatically, with no separate command. The switching conditions and step-by-step instructions for each mode of the v1 implementation are given in the “Migrating from the v1 implementation to the current one” section

below.

⚠ Starting from version 1.78 Deckhouse Platform no longer uses the v1 implementation of the module. The cluster switches to the current implementation after the upgrade to Deckhouse Platform 1.78, and the preparation for that is done before the upgrade, on Deckhouse Platform 1.77.

✖ Prepare your cluster in advance for the transition to the current implementation. The update to Deckhouse Platform 1.78, which no longer contains the v1 implementation, is blocked as long as the cluster is running in `Proxy` or `Local` mode of the v1 implementation, or in `Direct` mode without a configured `registry` in ModuleConfig.

You can find out which implementation of the module the cluster uses by checking whether the `registry-v2-switch` secret exists. To do this, follow the instructions.

Current implementation

In the current implementation, container image pull paths are configured through the `registry` ModuleConfig.

Modes of operation

In the current implementation, Deckhouse Platform supports the following container image pull path management modes (the mode is set with the `mode` parameter in the `registry` ModuleConfig):

- `Unmanaged` (the default mode). The `registry` module does not manage the pull paths of Deckhouse Platform component images: the cluster pulls images from the registry it was installed with (the registry is specified during cluster bootstrap in the `deckhouse` parameter in InitConfiguration).
- `Managed`. In this mode, the `registry` module manages the pull paths for container images. The following independent settings are used in this mode:
  - `primary.upstream` — the container image registry DP component images are pulled from. If the parameter is not set, the cluster is considered isolated (air-gapped). In an isolated cluster, the only source of container images is the in-cluster cache, which is filled with the `d8 mirror push` command.
  - `storage.cache` — controls the in-cluster cache on the master nodes. If the parameter is enabled, a container image registry is deployed on the master nodes, and all nodes pull images from it, with the upstream registry staying a fallback path while the cache is being filled.

The `primary.upstream` and `storage.cache` parameters together cover every supported configuration:

primary.upstream	storage.cache	What the cluster does
Set	`false`	Nodes pull images directly from the upstream registry. Nothing is deployed on the master nodes
Set	`true`	The cache acts as a pass-through: filled from the upstream registry on demand and ahead of time
Not set	`true`	The cluster is isolated (air-gapped). The cache is the only source. The `d8 mirror push` command is used to fill it
Not set	`false`	The configuration is rejected: nodes would have nowhere to pull images from

You can reconfigure cache enablement and disablement, change the image storage location, and change your credentials at any time. However, when removing an upstream registry to transition to an isolated state, the module waits until the entire expected set of images appears in the intra-cluster

cache—only then does it stop using the upstream registry. If the upstream registry stopped being used before the in-cluster cache held the whole expected image set, nodes could be left with no image source while the cache is being filled.

For more details on the architecture of the current implementation (the node agent, the in-cluster cache, leader replica election, how isolated clusters work), see the `registry` module documentation.

Installer behavior when no container image registry is configured

The installer creates the `registry` ModuleConfig with `mode: Managed` and `primary.upstream` pointing to the public Deckhouse Open/CE registry `registry.deckhouse.io/deckhouse/ce`, and warns about it in its output when the installation configuration specifies no container image registry in any of the following places:

- InitConfiguration
- The `registry` section of the `deckhouse` ModuleConfig
- The `registry` ModuleConfig (the ModuleConfig is absent)

This applies only when the installation configuration contains a ClusterConfiguration and the container runtime is managed by Deckhouse Platform (`defaultCRI` is `Containerd` or `ContainerdV2`). In a Managed Kubernetes cluster (without ClusterConfiguration) and with `defaultCRI: NotManaged`, the installer does not create this ModuleConfig.

Under the same conditions, the installer rejects a configuration in which the `registry` ModuleConfig sets `mode: Unmanaged` (or omits `mode`, which means the same) or `enabled: false` and no registry is specified elsewhere. In this case, the nodes would have nowhere to pull images from. Specify the registry from which nodes pull images directly in one of the following ways:

- Via the `deckhouse` parameter in InitConfiguration (`imagesRepo` and `registryDockerCfg`).
- By setting `mode: Managed` with `primary.upstream` in the `registry` ModuleConfig. A private or mirror registry populated with `d8 mirror push` can serve as `primary.upstream`. To install an air-gapped cluster from an image bundle instead, set `enabled: true` and `mode: Managed` with `storage.cache: true` and `storage.source`, without `primary.upstream`, and pass the bundle to the installer with the `--img-bundle-path` flag.

The installer also rejects a `registry` ModuleConfig that specifies `primary` or `storage` without `mode: Managed`, as the module schema does: in the default `Unmanaged` mode, these settings have no effect.

Garbage collection

Deckhouse Platform has a garbage collection mechanism for the container image registry. Each registry replica runs the operation on its own store. Garbage collection does not block image pulls from the store.

Garbage collection starts in the following cases:

- Once a day at night (at 3:17 a.m. Moscow Standard Time). Garbage collection is enabled by default. It is scheduled at night because filling the replica’s store is paused during the operation. A `d8 mirror push` command started during garbage collection can therefore fail.
- When all replicas have received a new image set: after a Deckhouse Platform update or after the `registry` module is enabled or updated.
- When the image store refuses writes because it has exceeded its designated storage limit or reached the reserved free space threshold on its filesystem.

For details about when garbage collection starts and what happens during the operation, see the documentation for the `registry` module.

You can also start garbage collection manually.

If you need to disable or configure garbage collection, use the `storage.garbageCollection` parameter in the `registry` ModuleConfig.

> ⚠ Disabling garbage collection only makes sense if the storage uses a disk with sufficient capacity to accommodate unlimited storage growth.

To view the status and schedule of garbage collection, use the commands below.

- List the recent collections — what started them, how they went, and how much they freed. A finished collection is kept for a day:

  ```
  d8 k get registryoperation -o wide
  ```

- View the time of the last garbage collection on each replica and any errors, if any:

```
d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq \
  'map({node, collectedAt, collectionError})'
```

- View the current garbage collection schedule:

```
d8 k get registrystorage registry -o jsonpath='{.spec.garbageCollection}' | jq
```

Garbage collection is the only mechanism that removes data from the registry. Each Deckhouse Platform release adds new images, so without garbage collection, the storage of a cluster that has been running for a long time will eventually fill up, and new images will no longer be added to it. However, for example, an isolated cluster with a full storage cannot be updated.

Collection removes the images of the releases the cluster has moved past. It keeps:

- the deployed release and the previous one, so that a rollback does not have to re-download anything;
- everything newer than the deployed release — an update in progress or, in an air-gapped cluster, a release pushed on purpose;
- every tag that is not a version: release channel names such as `stable`, floating tags, anything pushed by hand. The collector cannot know what these mean, so it does not touch them.

It also removes blobs that no image references, such as those left by an interrupted fill or by a collection whose cleanup failed. A collection removes them even when there are no images to delete.

The collector is deliberately cautious. In an isolated cluster, deleting a blob that is still needed is unrecoverable without another `d8 mirror push`, while keeping an unneeded one merely costs disk space. So a run that cannot determine what to keep — for example, when no deployed release is found — does nothing at all.

For more information about how the garbage collector works, see the `registry` module documentation.

Running garbage collection manually

> **i** Garbage collection created manually runs even when automatic collection is disabled.

Before starting garbage collection manually, assess whether it is needed. See the “Is a collection worth running now?” section of the `registry` module FAQ.

To start garbage collection manually, create a RegistryOperation object of type `GarbageCollection`:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: gc-manual-
spec:
  type: GarbageCollection
  cause: Free space before pushing a new release.
EOF
```

It runs even when automatic collection is disabled. Replicas collect one at a time, exactly as on schedule. A replica starts only when its store is not being written to: while the cache is being filled or a `d8 mirror push` is in progress, the replica waits, and the reason is shown in `status.replicas[].reason`. A store that refuses writes is not waited for: a fill cannot go on until the collection frees room. A replica still held back an hour after the operation was accepted fails with the reason `DeadlineExceeded`. An operation created while another one is running fails with the reason `AnotherOperationActive`.

Follow the operation and see what it removed:

```
d8 k get registryoperation <NAME> -o jsonpath='{.status}' | jq \
  '{phase, reason, message, garbageCollection}'
```

To cancel an operation, including one a replica is already running, set the cancel label on it:

```
d8 k label registryoperation <NAME> registry.deckhouse.io/cancel=true
```

What was deleted before the cancellation stays deleted.

If a replica's `syncer` container stops answering while the replica collects or waits for its turn, for example because its master went `NotReady`, the replica fails with the reason `ReplicaLost`, so that the operation does not hold every collection after it for a day:

- A replica that was collecting, once it has not held the Lease `registry-storage-gc` in the `d8-system` namespace for five minutes.
- A replica that was waiting, once its node has not been `Ready` for ten minutes. A master that is back sooner keeps its replica's turn.

A replica that waits on a node that stays `Ready` while its `syncer` does not answer is released by the cancel label, by the 24-hour timeout (the reason `TimedOut`), or by deleting the operation:

```
d8 k delete registryoperation <NAME>
```

Additional container image registries

`primary.upstream` is the only image registry configured in the `registry` ModuleConfig itself. It is used to pull DP component images. If you need additional registries (a vendor's, one needed by some module, or your own), add them using the RegistryUpstream custom resource.

An example of adding an additional registry is given in the "Adding an additional container image registry" section.

Requirements for using the current implementation

Before enabling container image pull path management with the current implementation of the `registry` module, make sure the cluster meets the following requirements:

- Nodes use containerd or containerd v2, set with the `defaultCRI` parameter in ClusterConfiguration.
- The cluster is fully managed by Deckhouse Platform. The current implementation of the module does not work in Managed Kubernetes clusters — use switching to a third-party registry instead.

Examples of configuring the current implementation

⚠ If a module's image wasn't pulled again and the module wasn't reinstalled while switching to the current implementation, follow the instructions to resolve the issue.

Configuring pull path management for Deckhouse Platform component images

To enable pull path management for DP component container images with the `registry` module, enable the module in `Managed` mode and specify the image registry details in `primary.upstream`. Example of the `registry` ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  version: 1
  enabled: true
  settings:
    mode: Managed
    primary:
      upstream:
        host: registry.deckhouse.io
        path: /deckhouse/ee
        scheme: HTTPS
        auth:
          license: <LICENSE_KEY> # Replace with your license key.
```

Instead of creating the ModuleConfig from scratch, you can use a ready-made one. The module publishes a ready-made configuration for your cluster, with the address, path, scheme, certificate authority, and credentials of the registry the cluster is already pulling images from.

To obtain a preconfigured ModuleConfig, run the following command:

```
d8 k -n d8-system get secret registry-suggested-config -o jsonpath='{.data.registry-mc\.yaml}'
| base64 -d
```

Review it and apply it — this avoids mistakes that are easy to make when retyping these values by hand (a truncated path, the wrong certificate authority, and so on).

To see how the change takes effect, use the commands below.

- View the state of the image registry configuration in Deckhouse Platform:

  ```
  d8 k get registryconfig registry -o jsonpath='{.status}' | jq
  ```

- Verify the implementation of the registry configuration on the nodes to ensure that everything has been successfully configured and to determine which backends are currently active:

  ```
  d8 k get registrynodes -o custom-columns=\
  NODE:.metadata.name,APPLIED:.status.observedGeneration,OK:.status.reconciled,BACKENDS:.status.activeBa
  ckends
  ```

Enabling the in-cluster cache

> **i** The in-cluster cache uses the disks of the master nodes and works in both static and cloud clusters managed by Deckhouse Platform. In a cloud cluster, that is the master node's root disk unless a separate disk is mounted at the cache directory (`/opt/deckhouse/registry`, or `/var/lib/deckhouse/registry` when the master NodeGroup has `systemType: DeckhouseEngine`). Size that disk for `storage.size` and the reserve the store keeps free for the node, on top of what the node itself uses. For details, see "Requirements for using the module" in the `registry` module documentation.

To enable the in-cluster cache, add the `storage.cache` setting to the `registry` ModuleConfig and specify the storage size:

```
spec:
  settings:
    mode: Managed
    primary:
      upstream:
        host: registry.deckhouse.io
        path: /deckhouse/ee
        auth:
          license: <LICENSE_KEY> # Replace with your license key.
    storage:
      cache: true
      size: 50Gi
```

Nothing on the nodes is reconfigured: the container runtime already turns to the agent for any registry, and the agent starts using the cache first. The upstream registry is used as a fallback path. So missing data in the cache right after the cache is enabled means a slower pull, not a failed one.

To check how full the in-cluster cache is, use the following command:

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq
'{phase,fill,leader,allReplicasFull}'
```

Turning the cache off is the same change in reverse, and just as safe. The blobs on disk are left untouched, and when the cache is turned on again, it is refilled from the data already accumulated. If you turn the in-cluster cache off and do not plan to turn it on again, remove the leftover cache data from the cluster nodes. For detailed instructions, see the `registry` module FAQ.

Switching the cluster into an air-gapped state

An isolated cluster has no upstream registry (it is not set in `primary.upstream`). For such a cluster, the cache is the only source of images. The `d8 mirror push` command is used to fill the storage. To verify that all required images are present in the cache, specify the expected set of images in the `storage.source` parameter

To move the cluster to air-gapped, follow these steps:

1. Pull the images on a machine that has internet access:

   ```
   d8 mirror pull --license <LICENSE_KEY> ./d8-bundle
   ```

2. Push them into the cluster through the publication endpoint:

   ```
   PUSH_SECRET=$(d8 k -n d8-system get secret registry-storage-push -o json)
   d8 mirror push ./d8-bundle registry.example.com/system/deckhouse \
     --registry-login "$(echo "$PUSH_SECRET" | jq -r .data.username | base64 -d)" \
     --registry-password "$(echo "$PUSH_SECRET" | jq -r .data.password | base64 -d)"
   ```

   The publication endpoint (the `registry` domain built from `publicDomainTemplate`, `registry.example.com` here) exists only while the cache is enabled, `publicDomainTemplate` is set and the `cert-manager` module is enabled. For details, see “Air-gapped clusters” in the `registry` module documentation.

3. Describe the expected image set in the cache and remove the `upstream` from the `registry` ModuleConfig:

   ```
   spec:
     settings:
       mode: Managed
       storage:
         cache: true
         size: 50Gi
         source:
           bundleRef: d8-mirror-bundle
           expectedDigests: 459
   ```

The upstream registry is removed from the nodes not at the moment the configuration is edited, but once the cache leader holds the whole expected image set — otherwise all nodes could be left with no image source. When the upstream is removed, the registry restarts, and the RegistryStorage resource may briefly report `Failed`: this is expected, wait for it to return to `Ready`. To check the status of the transition, use the following commands.

Check whether the cache leader holds the whole expected image set:

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq '{safeToDropUpstream,fill}'
```

Check whether the cluster is isolated (while `effectiveUpstream` is set, the cluster is using it; if it is empty, the cluster is isolated):

```
d8 k get registryconfig registry -o jsonpath='{.status.effectiveUpstream}' | jq
```

Adding an additional container image registry

> **i** Requests to additional registries are always routed through the node agent (credentials and the certificate authority are kept in one place), and images from them are not cached.

To add an additional image registry (not for DP system components), use the RegistryUpstream resource. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryUpstream
metadata:
  name: virtualization-images
spec:
  match: images.virtualization.example.com
  upstream:
    host: vendor.example.com
    path: /virtualization
    auth:
      username: robot
      password: <PASSWORD>
```

After creating the RegistryUpstream, check that it has been accepted. The object is rejected in the following cases:

- It conflicts with the primary registry or another object claiming the same `match` value. The object is rejected rather than merged.
- It contains an invalid field, such as a `ca` that is not a PEM certificate.
- Adding the object exceeds either of the following limits:
  - Settings from no more than 256 RegistryUpstream objects can be applied at a time.
  - The combined size of the accepted objects' `spec` fields in JSON format must not exceed 256 KiB.

To check whether the object has been accepted, run the following command:

```
d8 k get registryupstreams -o custom-columns=\
NAME:.metadata.name,MATCH:.spec.match,ACCEPTED:.status.conditions[0].status,REASON:.status.conditions[0].rea
son
```

In the example above, after the additional registry is added, image pull requests for `images.virtualization.example.com` are routed by the agent on every node to `vendor.example.com/virtualization`, with the credentials and certificate authority held by the cluster rather than by each workload individually. Nothing on the nodes is reconfigured for this.

Pulling from a private container image registry without declaring it

To load images from a storage module that is unknown to the module, no additional declarations are required. The node agent proxies such requests untouched, along with whatever credentials the request already carried. So an ordinary `imagePullSecret` works exactly as it does in a cluster where the module was never enabled.

To create a secret with the credentials of the private container image registry, run:

```
d8 k create secret docker-registry my-private-registry \
  --docker-server=private.example.com \
  --docker-username=robot \
  --docker-password=<PASSWORD>
```

To use this secret when pulling images, specify it in the pod's `imagePullSecrets`:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  imagePullSecrets:
  - name: my-private-registry
  containers:
  - name: app
    image: private.example.com/team/app:v1
```

It is worth creating a RegistryUpstream for the registry only if you want the cluster to hold the credentials instead of every workload doing so individually, or if the registry needs a certificate authority the nodes do not have.

Turning off pull path management

To turn off pull path management, set `mode: Unmanaged` in the `registry` ModuleConfig:

```
spec:
  settings:
    mode: Unmanaged
```

After `mode: Unmanaged` is set, pull path management through the `registry` module is turned off. The cluster resumes downloading images from the container registry specified in the `deckhouse-registry` secret—that is, from the same source from which images were downloaded before the module was enabled. Cache data on the master nodes is deliberately preserved: when pull path management is turned on again, the cache is topped up only with the images missing from the container image registry rather than filled from scratch. If the cache is no longer needed, free up the space it occupies by following the instructions in the `registry` module documentation.

V1 implementation

> ⚠ The v1 implementation of the `registry` module described below was used in Deckhouse Platform up to and including 1.77. Deckhouse Platform 1.78 no longer contains it. For new clusters, and for existing ones at the first opportunity, use the current implementation.

In the v1 implementation, Deckhouse Platform supports several management modes for the platform component registry settings: `Unmanaged`, `Direct`, `Proxy`, and `Local`. In `Direct`, `Proxy`, and `Local` modes, a fixed virtual address is used to access the Deckhouse Platform component registry. This eliminates the need to restart all control plane components and re-download images when the platform component registry changes.

In `Unmanaged` mode, no virtual address is used. The external registry is accessed directly: if the registry changes, all Deckhouse Platform components are restarted.

Switching between modes and registries is done via the `deckhouse` ModuleConfig. The switch occurs automatically (see the switching examples below for more details).

The architecture of the v1 implementation's modes is described in the "Registry module" section.

Features of the Deckhouse Platform component registry settings management modes:

- `Direct`: Provides direct access to an **external** registry via the fixed virtual address `registry.d8-system.svc:5001/system/deckhouse`. The fixed address prevents images from being re-downloaded and components from being restarted when registry parameters change.

- `Proxy`: Uses an **internal caching proxy registry** that accesses an **external** registry. The caching proxy registry runs on control-plane (master) nodes. This mode reduces the number of requests to the external registry by caching images. The internal registry is accessed via the fixed virtual address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` mode.
- `Local`: Uses a local **internal** registry, with the registry running on control-plane (master) nodes. This mode allows the cluster to operate in an isolated environment. The internal registry is accessed via the fixed virtual address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` and `Proxy` modes.
- `Unmanaged` (configurable mode): Works without an internal registry. No virtual address is used. The **external** registry is accessed directly.

⚠ This document refers to the configurable `Unmanaged` mode (of the v1 implementation of the `registry` module) and the non-configurable `Unmanaged` mode (the fully deprecated format, without the `registry` module).

The non-configurable `Unmanaged` mode does not use the `registry` module at all (for example, in Managed Kubernetes clusters). The Deckhouse Platform component registry configuration parameters are set during cluster installation or, in a deployed cluster, using the `helper change registry` utility (deprecated).

Restrictions on Deckhouse Platform component registry settings

There are a number of restrictions and considerations regarding cluster installation, the v1 implementation's usage conditions, and switching between its modes.

Restrictions during cluster installation

The following restrictions apply when installing a cluster:

- A new cluster cannot be installed with the v1 implementation: the installer refuses `Proxy` and `Local` in the `deckhouse` ModuleConfig. A `Direct` section there is read only as the registry address, and the cluster is installed in `Unmanaged` mode. To install a cluster whose pull path the `registry` module manages, use the current implementation.
- To launch a cluster in the non-configurable `Unmanaged` mode (Legacy, without using the `registry` module), registry parameters must be specified in InitConfiguration.

Operating conditions restrictions

To manage Deckhouse Platform component registry settings using the v1 implementation, the following conditions must be met:

- CRI containerd or containerd v2 must be used on cluster nodes. To configure CRI, refer to the ClusterConfiguration configuration.
- The cluster must be fully managed by Deckhouse Platform. In Managed Kubernetes clusters, managing Deckhouse Platform component registry settings via the `registry` module is not available.

Restrictions on mode switching

Mode switching restrictions are as follows:

- Changing registry parameters and switching modes is only available after the bootstrap phase is fully complete.
- For the first switch, you must migrate user registry configurations. For more details, see the "Registry Module: FAQ" section.
- Switching to the non-configurable `Unmanaged` mode (without using the `registry` module) is only available from the configurable `Unmanaged` mode. For more details, see the "Registry Module: FAQ" section.

Examples of switching between the v1 implementation's modes

⚠ If a module's image wasn't pulled and the module wasn't reinstalled during the switch, follow the instructions to resolve the issue.

Switching to Unmanaged mode

To switch an already running cluster to `Unmanaged` mode, follow these steps:

✖ Changing the registry in `Unmanaged` mode will restart all Deckhouse Platform components.

1. Make sure the `registry` module is enabled and running. To do this, execute the following command:

   ```
   d8 k get module registry -o wide
   ```

   Example output:

   ```
   NAME       WEIGHT ...  PHASE   ENABLED   DISABLED MESSAGE   READY
   registry   38     ...  Ready   True                         True
   ```

2. Ensure the Deckhouse Platform queue is empty and has no errors:

   ```
   d8 system queue list
   ```

   Example output:

   ```
   Summary:
   - 'main' queue: empty.
   - 107 other queues (0 active, 107 empty): 0 tasks.
   - no tasks to handle.
   ```

3. Set the `Unmanaged` mode settings in the ModuleConfig `deckhouse`. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module configuration for correct setup.

   Configuration example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
         unmanaged:
           imagesRepo: registry.deckhouse.io/deckhouse/ee
           scheme: HTTPS
           license: <LICENSE_KEY> # Replace with your license key.
   ```

4. Check the registry mode switch status in the `registry-state` secret using the instruction.

   Example output:

   ```
   conditions:
   # ...
     - lastTransitionTime: "..."
       message: ""
       reason: ""
       status: "True"
       type: Ready
   hash: ..
   mode: Unmanaged
   target_mode: Unmanaged
   ```

5. If you need to disable automatic Deckhouse Platform updates from the configured image registry, remove the `releaseChannel` parameter from the `deckhouse` module configuration. After that, automatic platform updates will be disabled, and you will need to manage the Deckhouse Platform version manually.

If you need to switch to the legacy registry management method (without the `registry` module), see the instruction.

⚠ Managing the Deckhouse Platform component image registry without the `registry` module is a deprecated management format.

Migrating from the v1 implementation to the current one

This section describes the transition from the v1 implementation of the `registry` module to the current one. A cluster that does not use the `registry` module at all (the fully deprecated format) has nothing to hand over, since the v1 implementation never configured its nodes: it needs no preparation for the upgrade to Deckhouse Platform 1.78 and keeps pulling images from the registry it was installed with.

The migration is the handover of the pull path from the v1 implementation to the current one. It does not need to be started manually, and there is no separate command for it: the `registry` module takes over control automatically after the cluster is upgraded to Deckhouse Platform 1.78. The only condition is that by then the v1 implementation must have released the pull path. Both implementations configure the same thing on every node — which container image registry the container runtime pulls images from and with which credentials — so they never manage a cluster at the same time.

Determining the migration method and steps

What needs to be done to hand over pull path management to the current implementation of the `registry` module, and when, depends on the mode the v1 implementation is running in. The mode is set in the `settings.registry.mode` parameter of the `deckhouse` ModuleConfig. To view the mode, use the following command:

```
d8 k get mc deckhouse -o jsonpath='{.spec.settings.registry.mode}'
```

Empty output means `Unmanaged` mode: the registry settings of this cluster have never been set. Such a cluster needs no preliminary actions for the migration — the handover happens automatically.

The actions required to migrate from the v1 implementation to the current one are described in the table:

Mode of the v1 implementation	What to do	When
`Unmanaged`	Nothing — the handover happens on its own	—
`Direct`	Configure the `registry` ModuleConfig: set `mode: Managed` and `primary.upstream`	Before upgrading to Deckhouse Platform 1.78
`Proxy`	Switch the cluster to `Unmanaged`	Before upgrading to Deckhouse Platform 1.78
`Local`	Follow the separate procedure for air-gapped clusters	Before upgrading to Deckhouse Platform 1.78

⚠ Deckhouse Platform 1.78 no longer contains the v1 implementation of the module. The cluster switches to the current implementation after the upgrade to Deckhouse Platform 1.78, and the preparation for that is done before the upgrade, on Deckhouse Platform 1.77.

> ✖ Prepare your cluster in advance for the transition to the current implementation. The update to Deckhouse Platform 1.78, which no longer contains the v1 implementation, is blocked as long as the cluster is running in `Proxy` or `Local` mode of the v1 implementation, or in `Direct` mode without a configured `registry` in ModuleConfig.

When the v1 implementation is used in `Direct` mode, no mode switch is needed. The `registry` ModuleConfig settings are deliberately accepted already in Deckhouse Platform 1.77: written before the upgrade, they are stored without effect and take effect on the module's first reconciliation after the upgrade.

Checking which implementation the cluster uses

When the `registry` module takes over pull path management, it records that in the `registry-v2-switch` secret. Check whether the secret exists:

```
d8 k -n d8-system get secret registry-v2-switch >/dev/null 2>&1 \
  && echo "current implementation" || echo "v1 implementation"
```

If the cluster is still on the v1 implementation, the module reports the reason on every reconciliation, and the `D8RegistryMigrationPending` alert fires in the cluster. To see which actions are required, use the following command:

```
d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
```

Migrating from Unmanaged mode

In `Unmanaged` mode, the v1 implementation does not manage the pull path, so no preliminary preparation for switching to the current implementation is required — the handover happens automatically.

The migration procedure to the current implementation in this case is as follows:

1. If the cluster on the v1 implementation is being switched to `Unmanaged` from another mode, wait for the transition to complete. The status must show `mode: Unmanaged` with no pending target mode.

   To check, use the following command:

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

2. Upgrade the cluster to Deckhouse Platform 1.78 (or, if it has already been upgraded, wait for the next reconciliation). The module takes over image pull path management automatically, and the pull source does not change. The default mode of the `registry` module in the current implementation is also `Unmanaged`, so the cluster keeps pulling images from the same registry as before. On containerd v1 nodes, however, the takeover restarts containerd node by node if the v1 implementation left a node registry configuration, and enabling `mode: Managed` in the next step restarts it (a second time, if the takeover already did; for details, see "Does the migration restart containerd?").

3. To have the module manage the pull paths, set `mode: Managed` in the `registry` ModuleConfig and specify the registry to pull images from. A ready-made configuration for your cluster is published in the `registry-suggested-config` secret. For a configuration example, see the "Configuring pull path management for Deckhouse Platform component images" section.

Migrating from Direct mode

In `Direct` mode of the v1 implementation, nodes pull images through an in-cluster address served by the v1 implementation's proxy. The current implementation serves the same address, so the migration is a direct handover of the address: no switch through `Unmanaged` is needed, and no components are restarted.

The migration procedure to the current implementation in this case is as follows:

1. Configure the `registry` ModuleConfig before the upgrade — without this configuration the upgrade is blocked. To fill in the `registry` ModuleConfig, take the values from the `registry.direct` section of the `deckhouse` ModuleConfig:
   - split the `imagesRepo` value from the `deckhouse` ModuleConfig into `host` and `path` and

specify them in the corresponding fields of the `registry` ModuleConfig;
- use the credentials specified in the `deckhouse` ModuleConfig: the same `license` (or `username`/`password`), and `ca` if the registry requires one.

Example of the `registry` ModuleConfig for the switch:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: true
  version: 1
  settings:
    mode: Managed
    primary:
      upstream:
        scheme: HTTPS
        host: registry.deckhouse.io
        path: /deckhouse/ee
        auth:
          license: <LICENSE_KEY> # Replace with your license key.
```

The v1 implementation stores these settings but does not act on them, so nothing changes in the cluster until the upgrade.

2. Upgrade the cluster to Deckhouse Platform 1.78. The handover of container image pull path management happens on the next reconciliation, and image pulls keep working throughout: the Service and the proxy of the v1 implementation keep serving the in-cluster address until the agent of the current implementation takes it over on every node, and only then are they removed.
3. Track the progress of the handover using the following commands.
   - Check whether path management has been handed over to the current implementation:

     ```
     d8 k -n d8-system get secret registry-v2-switch >/dev/null 2>&1 && echo "handed over"
     ```

   - Check the state of the image registry nodes:

     ```
     d8 k get registrynode -o custom-columns='NODE:.metadata.name,READY:.status.reconciled,SERVING:.status.proxyListening'
     ```

   The handover is recorded a few minutes after the new version starts, the agent appears on the nodes a few minutes later, and the objects of the v1 implementation are removed shortly after that. Image pulls keep working at each of these stages.

Migrating from Proxy mode

When the v1 implementation is used in `Proxy` mode, a proxy of its own, with its own certificates, is deployed on every node. The current implementation cannot adopt this state. So the cluster must first be switched to `Unmanaged`, and this can only be done before the upgrade.

The migration procedure to the current implementation in this case is as follows:

1. In the `deckhouse` ModuleConfig, set `registry.mode: Unmanaged`, keeping the same registry address and credentials (for an example of the `deckhouse` ModuleConfig, see the “Switching to Unmanaged mode” section). All nodes are reconfigured to pull directly from the external registry, so the caching provided in `Proxy` mode is lost until the last step.
2. Wait for the transition to complete — `mode: Unmanaged` with no pending target mode. A cluster caught mid-transition cannot be migrated; the status shows which mode it is still switching to:

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

3. Upgrade the cluster to Deckhouse Platform 1.78. The handover happens on the next reconciliation, and the pull source does not change: in `Unmanaged` mode, the current

implementation does not manage the pull path either. On containerd v1 nodes, however, the takeover restarts containerd node by node, and enabling `mode: Managed` in the next step restarts it again (for details, see “Does the migration restart containerd?”).

4. To get in-cluster caching back, set `mode: Managed` with `storage.cache: true` and the same external registry. Note that the cache of the current implementation is built differently from `Proxy` mode of the v1 implementation: a single store with replicas on the master nodes instead of a proxy on every node. Before enabling it on a cluster with little free disk space on the master nodes, read how the cache is filled and cleaned up.

Migrating an air-gapped cluster from Local mode

In `Local` mode of the v1 implementation, the cluster has no external image registry — its role is played by a registry inside the cluster itself. So the standard migration procedure does not work here, since it goes through `Unmanaged` mode, where every node pulls images directly from an external registry, and such a cluster has nowhere to pull them from.

An external registry has to be connected to the cluster for the duration of the migration. It runs in the same cluster, but outside the Deckhouse Platform namespaces. The cluster is switched to it and upgraded, and once the registry of the current implementation is filled with images, the temporary registry is removed.

> ⚠ Before starting, make sure the master nodes have free disk space for **four image sets**: at the peak of the migration, three copies of the set exist at the same time — the `Local` store, the temporary registry, and the registry of the current implementation being filled — and the space for the fourth set is headroom the node must not run out of. The detailed disk space calculation and the verification procedure are described in the “Disk requirements” section of the `registry` module documentation.

Migration procedure:

1. Run a temporary external registry in its own namespace, outside Deckhouse Platform management. Any container image registry implementation will do, but it must serve TLS with a certificate the cluster can verify, and Deckhouse Platform must not manage it: whatever happens to the module’s objects, the temporary container image registry must keep working.

   In addition, the container image registry must be reachable **at the same address from two places**: in step 3, nodes pull images from it, and in step 6, the `registry` module’s syncer running in a pod reads it. The certificate must cover the chosen address. Options tested on a single cluster:

Address	From a node	From a pod
A `hostNetwork` port on the node’s own IP (`<NODE_IP>:5000`)	works	works
A Service name (`<SERVICE>.<NAMESPACE>.svc:<PORT>`)	does not resolve	works
A NodePort on the node’s IP	works	does not work: `operation not permitted`

   Use the first option: run the container image registry with `hostNetwork: true` on one node, address it by that node’s IP, and add that IP to the certificate’s SAN — then the whole migration goes through a single address. A Service name looks neater, but stops working in step 3, because the node’s container runtime does not resolve names through the cluster DNS.

2. Load the image set of the current version into the temporary registry: `d8 mirror pull` on a machine with internet access, then `d8 mirror push` into the temporary registry.
3. In the `deckhouse` ModuleConfig, set `registry.mode: Unmanaged` with the address, CA certificate, and credentials of the temporary registry. The data of the `Local` store stays where it is, in the `/opt/deckhouse/registry` directory on the master nodes.
4. Make sure the transition is complete and images are really pulled from the temporary registry (`mode: Unmanaged` with no pending target mode in the `registry-state` secret):

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

5. Upgrade the cluster to Deckhouse Platform 1.78. The handover happens on the next reconciliation, and the cluster keeps pulling images from the temporary registry throughout. On

containerd v1 nodes the takeover restarts containerd node by node, and enabling `mode: Managed` in the next step restarts it again (see Does the migration restart containerd?).

6. Enable the current implementation: set `mode: Managed` with `storage.cache: true`, specify the temporary registry as `primary.upstream` — **and add `storage.source` in the same edit** (without it, the next step is rejected). The registry of the current implementation uses the same host path that `Local` used, so the images already on disk are not downloaded again.

7. Wait until the storage reports that it holds the whole image set (`phase: Ready` and `safeToDropUpstream: true`), and remove `primary.upstream` from the `registry` ModuleConfig. The cluster is isolated again — now on the current implementation:

   ```
   d8 k get registrystorage registry -o jsonpath='{.status.phase}
   {.status.safeToDropUpstream}{"\n"}'
   ```

   Removing the upstream restarts the registry, and the RegistryStorage resource may briefly report `Failed`: this is expected, wait for it to return to `Ready`.

8. Delete the temporary registry and reclaim its disk space.

Migration to registry management format using the registry module

> **i** This procedure applies to Deckhouse Platform 1.77 and earlier only: it moves a cluster from the deprecated format (without the `registry` module, configured only via `InitConfiguration`) to the **v1 implementation** of the module, which Deckhouse Platform 1.78 no longer contains. It is not a step of the migration to the current implementation: a cluster in the deprecated format needs no preparation for the upgrade to Deckhouse Platform 1.78. See the “Migrating from the v1 implementation to the current one” section.

During the migration, containerd v1 will be switched to the new registry configuration scheme. containerd v2 uses the new scheme by default. For more details, see the section describing configuration methods.

To view the container runtime type used by default on cluster nodes (in NodeGroups), use the following command:

```
d8 system edit cluster-configuration
```

The container runtime type is specified in the `defaultCRI` parameter.

If containerd v2 is used as the container runtime, follow the containerd v2 migration instructions.

If containerd v1 is used as the container runtime, you can choose one of the following methods to migrate to the `registry` module’s image registry management format:

- Without switching to containerd v2. In this case, follow the instructions for containerd v1.
- With switching to containerd v2. In this case:
  1. Verify whether you can switch to containerd v2, and if so, perform the switch.
  2. After migrating, follow the instructions for containerd v2.

For containerd v2

1. Switch to using the `registry` module. To do this, specify the `Unmanaged` mode parameters in the `deckhouse` ModuleConfig. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module configuration for correct setup.

   You can view the current registry settings using the following command:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller global
   values | yq e '.modulesImages.registry' -
   ```

   Specify these settings when configuring the `Unmanaged` mode:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Replace with your license key.
```

2. Wait for the switch to complete. Example status output:

```
conditions:
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

If you need to add configurations for an additional image repository, refer to the section “New way to add configuration for an additional registry”.

For containerd v1

- During the switch, containerd v1 will be restarted.
- During the switch, containerd v1 will be migrated to the new Deckhouse Platform component registry configuration scheme.
- During the switch, custom registry configurations for containerd v1 will be temporarily unavailable.

1. Make sure that nodes with containerd v1 do not have any custom registry configurations located in the `/etc/containerd/conf.d` directory.
2. If configurations are present, you need to migrate to the new registry configuration format in containerd. To do this, add new configuration files to the `/etc/containerd/registry.d` directory. These configurations will take effect after switching to the `registry` module. To add configurations, prepare a NodeGroupConfiguration. For more details, see the section with a description of configuration methods. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not
required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - <<
EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        [host."https://${REGISTRY_URL}".auth]
          username = "username"
          password = "password"
    EOF
```

3. Apply the NodeGroupConfiguration. Wait until the configuration files appear in the `/etc/containerd/registry.d` directory on all nodes.
4. Verify that the configurations are working correctly. To do this, use the following command:

```
# For HTTPS.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/registry/path:tag

# For HTTP.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/registry/path:tag
```

5. Switch to using the `registry` module. To do this, specify the `Unmanaged` mode parameters in the `deckhouse` ModuleConfig. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module configuration for correct setup.

   You can view the current registry settings using the following command:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller global
   values | yq e '.modulesImages.registry' -
   ```

   Specify these settings when configuring the `Unmanaged` mode:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Replace with your license key.
```

6. After applying, wait for the following message in the switch status:

   Example output:

   ```
   conditions:
   # ...
   - lastTransitionTime: "2025-08-13T15:22:34Z"
     message: |
       Check current nodes configuration
       2/2 node(s) Unready:
       - master-0: has custom toml merge containerd configuration
       - worker-5e389be0-578df-s5sm5: has custom toml merge containerd
   configuration
     reason: Processing
     status: "False"
     type: ContainerdConfigPreflightReady
   ```

   This message means that there are old registry configurations on the nodes located in the `/etc/containerd/conf.d` directory. The switch to the new containerd configuration is currently blocked. To allow the switch, you need to remove the old configuration files.

7. Remove the old configuration files to allow switching to the `registry` module. To do this, create a NodeGroupConfiguration. Example of a NodeGroupConfiguration manifest:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth-delete.sh
spec:
  # This step must be completed before '032_configure_containerd.sh'.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    file="/etc/containerd/conf.d/old-config.toml"

    [ -f "$file" ] && rm -f "$file"
```

8. After removing the old configurations, make sure that the switch has resumed. Example of the switch status:

```
conditions:
# ...
- lastTransitionTime: "2025-08-13T16:42:09Z"
  message: ""
  reason: ""
  status: "True"
  type: ContainerdConfigPreflightReady
```

9. Wait for the switch to complete. Example of the switch status:

```
conditions:
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

10. Delete the NodeGroupConfiguration created in the step for deleting old configuration files:

```
d8 k delete nodegroupconfiguration containerd-additional-config-auth-delete.sh
```

To verify that NodeGroupConfiguration has been deleted, use the command:

```
d8 k get nodegroupconfiguration
```

The list should not contain the NodeGroupConfiguration to be deleted (for this example, `containerd-additional-config-auth-delete.sh`).

Migration to the deprecated registry management format (without the registry module)

✖
- This is a deprecated Deckhouse Platform component registry management format.
- During the switch, containerd v1 will be restarted.
- During the switch, containerd v1 will be migrated to the legacy registry configuration scheme.
- During the switch, custom registry configurations for containerd v1 will be temporarily unavailable.

1. Switch the `registry` module to configurable `Unmanaged` mode. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module configuration for correct setup.

   Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
         unmanaged:
           imagesRepo: registry.deckhouse.io/deckhouse/ee
           scheme: HTTPS
           license: <LICENSE_KEY> # Replace with your license key.
   ```

2. Check the switch status using the instruction. Example output:

   ```
   conditions:
   # ...
   - lastTransitionTime: "..."
     message: ""
     reason: ""
     status: "True"
     type: Ready
   hash: ..
   mode: Unmanaged
   target_mode: Unmanaged
   ```

3. Switch the registry to the non-configurable `Unmanaged` mode (without `registry` module). Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
   ```

4. Check the switch status using the instruction. Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

5. If containerd v1 is used and custom registry configurations are applied in the cluster, they must be replaced with the old format. To do this, prepare the registry configurations in the old format. These configurations do not need to be applied at this stage. Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
            endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
            username = "username"
            password = "password"
            # OR
            auth = "<BASE64_AUTH_STRING>"
    EOF
```

6. Delete the `registry-bashible-config` secret. This will trigger containerd v1 to switch back to

the legacy registry format:

```
d8 k -n d8-system delete secret registry-bashible-config
```

7. After deletion, wait for the switch to complete. Use the instruction to track the progress. Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

8. If containerd v1 is used, apply the previously prepared NodeGroupConfiguration with custom registry configurations.
9. Disable the `registry` module. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: false
  settings: {}
  version: 1
```

Checking the registry mode switch status in the v1 implementation

The status of the Deckhouse Platform component registry mode switch in the v1 implementation can be retrieved using the following command:

```
d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
map_values(@base64d) | .conditions = (.conditions | from_yaml)'
```

Example output:

```
conditions:
  - lastTransitionTime: "2025-07-15T12:52:46Z"
    message: 'registry.deckhouse.io: all 157 items are checked'
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
  - lastTransitionTime: "2025-07-11T11:59:03Z"
    message: ""
    reason: ""
    status: "True"
    type: ContainerdConfigPreflightReady
  - lastTransitionTime: "2025-07-15T12:47:47Z"
    message: ""
    reason: ""
    status: "True"
    type: TransitionContainerdConfigReady
  - lastTransitionTime: "2025-07-15T12:52:48Z"
    message: ""
    reason: ""
    status: "True"
    type: InClusterProxyReady
  - lastTransitionTime: "2025-07-15T12:54:53Z"
    message: ""
    reason: ""
    status: "True"
    type: DeckhouseRegistrySwitchReady
  - lastTransitionTime: "2025-07-15T12:55:48Z"
    message: ""
    reason: ""
    status: "True"
    type: FinalContainerdConfigReady
  - lastTransitionTime: "2025-07-15T12:55:48Z"
    message: ""
    reason: ""
    status: "True"
    type: Ready
mode: Direct
target_mode: Direct
```

The output displays the status of the switch process. Each condition can have a status of `True` or `False`, and may contain a `message` field with additional details.

Description of conditions:

Condition	Description
`ContainerdConfigPreflightReady`	State of the containerd configuration preflight check. Verifies there are no custom containerd auth configurations on the nodes
`TransitionContainerdConfigReady`	State of preparing the containerd configuration for the new mode. Verifies that the configuration contains both the old and new mode settings
`FinalContainerdConfigReady`	State of finalizing the switch to the new containerd mode. Verifies that the containerd configuration has been successfully applied and contains only the new mode settings
`DeckhouseRegistrySwitchReady`	State of switching Deckhouse Platform and its components to use the new registry. `True` means Deckhouse Platform successfully switched and is ready to operate

Condition	Description
`InClusterProxyReady`	State of In-Cluster Proxy readiness. Checks that the In-Cluster Proxy has started successfully and is running
`CleanupInClusterProxy`	State of cleaning up the In-Cluster Proxy if it is not needed in the selected mode. Verifies that all related resources have been removed
`NodeServicesReady`	State of Node Services Manager and Static-Pod registry readiness. Verifies that the Node Services Manager is successfully launched and operational, and that the Static-Pod registry has been successfully deployed by it
`CleanupNodeServices`	State of cleaning up the Node Services Manager and Static-Pod registry if they are not needed in the selected mode. Verifies that all related resources have been removed
`RegistryContainsRequiredImages`	State of checking the registry for the presence of required images
`Ready`	Overall state of registry readiness in the selected mode. Indicates that all other conditions are met and the module is ready to operate

Additional resources

- Global custom resources
- Global parameters
- Global custom resources
- Module `deckhouse` documentation
- Module `node-manager` documentation
- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Switching a Managed Kubernetes cluster to use third-party registry

> ⚠ This page applies only to Managed Kubernetes clusters. Registry management through the `registry` module is not available in these clusters.
>
> For clusters fully managed by Deckhouse Platform (DP), registry settings are configured using the `registry` module (or, in clusters still running its previous implementation, in the `registry` section of the `deckhouse` module configuration). For configuration examples and descriptions of the implementations, see “Managing the registry in DP-managed clusters”.
>
> The information on this page applies to clusters that use containerd v1 or containerd v2 as their primary container runtime. However, containerd v2 is the current standard for DP. If you need to migrate to containerd v2, refer to the section “Migrating container runtime to containerd v2” to verify availability and perform the migration.

> The methods for adding a configuration for a registry are described in the section “Adding a configuration for an additional registry”.

To switch the cluster to use an external registry, follow these steps:

1. Run the `deckhouse-controller helper change-registry` command from the DP pod with the

parameters of the new registry.
Example:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry --user MY-USER --password MY-PASSWORD registry.example.com/deckhouse/ee
```

2. If the registry uses self-signed certificates, place the corresponding root certificate in the file `/tmp/ca.crt` inside the DP pod and add the `--ca-file /tmp/ca.crt` option to the command. Alternatively, insert the CA content into a variable, as shown below:

```
CA_CONTENT=$(cat <<EOF
-----BEGIN CERTIFICATE-----
CERTIFICATE
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
CERTIFICATE
-----END CERTIFICATE-----
EOF
)
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- bash -c "echo '$CA_CONTENT' >
/tmp/ca.crt && deckhouse-controller helper change-registry --ca-file
/tmp/ca.crt --user MY-USER --password MY-PASSWORD
registry.example.com/deckhouse/ee"
```

   To view the list of available flags for the `deckhouse-controller helper change-registry` command, run:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry --help
```

   Example output:

```
usage: deckhouse-controller helper change-registry [<flags>] <new-registry>
Change registry for deckhouse images.
Flags:
  --help               Show context-sensitive help (also try --help-long and --
help-man).
  --user=USER          User with pull access to registry.
  --password=PASSWORD  Password/token for registry user.
  --ca-file=CA-FILE    Path to registry CA.
  --scheme=SCHEME      Used scheme while connecting to registry, http or https.
  --dry-run            Don't change deckhouse resources, only print them.
  --new-deckhouse-tag=NEW-DECKHOUSE-TAG
                       New tag that will be used for deckhouse deployment image
(by default
                       current tag from deckhouse deployment will be used).
Args:
  <new-registry>  Registry that will be used for deckhouse images (example:
                  registry.deckhouse.io/deckhouse/ce). By default, https will
be used, if you need
                  http - provide '--scheme' flag with http value
```

3. Wait until the `registry` Pod reaches the `Ready` status. If the Pod is in the `ImagePullBackoff` state, restart it.

4. Wait for bashible to apply the new settings on the master node.

   Check the bashible system service log on the master node, for example, using the following command:

```
journalctl -u bashible -n 20
```

   The log should contain the message `Configuration is in sync, nothing to do`.

Example of output when viewing the bashible service log:

```
Aug 13 05:03:08 kube-master-0 systemd[1]: Started Bashible service.
Aug 13 05:03:10 kube-master-0 bash[1847265]: Configuration is in sync, nothing
to do.   <--
Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Deactivated
successfully.
Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Consumed 1.075s CPU
time.
```

5. If you need to disable automatic DP upgrades through the external registry, remove the `releaseChannel` parameter from the `deckhouse` module configuration. After that, automatic platform upgrades will be disabled, and you will need to manage the DP version manually.
6. Check if any pods in the cluster are still using the original registry address:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[]
  | select(.image | startswith("registry.deckhouse"))) | .metadata.namespace +
"\t" + .metadata.name' | sort | uniq
```

⚠ If a module's image wasn't pulled and the module wasn't reinstalled during the switch, follow the instructions to resolve the issue.

Additional resources

- Module `deckhouse` parameters
- Module `registry` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Authentication

Authentication is the process of verifying a user's identity, providing access control to all interfaces of the Deckhouse Platform (DP) and cluster resources.
The platform implements end-to-end authentication, allowing a unified mechanism to be applied both to internal components and user applications.

The core of the authentication mechanism is a federated OpenID Connect (OIDC) provider — `Dex`.
Learn more about how authentication works in the Architecture section.

Depending on the configuration, DP supports two authentication approaches:

- Local authentication — users and groups are created directly in the cluster and stored as User and Group resources.
  The User resource stores a hashed version of the password (bcrypt), not the plain-text password.
- Integration with external providers — enables connection to systems like LDAP, GitLab, GitHub, and others to support single sign-on across multiple DP clusters.

From the perspective of a cluster user or application developer, the method chosen by the administrator to configure authentication in DP does not matter — the authentication interface and integration steps are the same. You can learn about configuring authentication for applications in the section Usage → IAM → Authentication.

The platform also provides capabilities for:

- Enabling authentication for any web application in the cluster.
- Configuring authenticated access to the Kubernetes API via a load balancer.

Additional resources

- Module `user-authn` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with external authentication providers

Connecting an external authentication provider allows you to use a single set of credentials to access multiple clusters and simultaneously work with multiple providers.

DP supports integration with the following external authentication providers and protocols:

- LDAP (for example, Active Directory);
- OIDC (for example, Okta, Keycloak, Gluu, Blitz Identity Provider);
- SAML 2.0;
- GitHub integration;
- GitLab integration;
- Atlassian Crowd integration;
- Bitbucket Cloud integration.

> Password policies (such as complexity requirements, expiration, history, two-factor authentication, etc.) are fully controlled by the external authentication provider.
> DP does not manage passwords or interfere with policy enforcement on the provider side.

General integration workflow

> The `allowedGroups` parameter in the DexProvider resource allows you to restrict login access to users who belong to specific groups.
> If the `allowedGroups` list is specified, the user **must** be a member of at least one of these groups — otherwise, authentication will be considered unsuccessful.
> If the parameter is not specified, no group-based filtering will be applied.

1. Create an OAuth application in the authentication provider:
   - Set the redirect URI to `https://dex.<publicDomainTemplate>/callback`.
   - Obtain the `clientID` and `clientSecret`.

   > **Important**: When specifying the redirect URI, substitute the actual value of `publicDomainTemplate` without `%s`.
   > For example, if `publicDomainTemplate: '%s.sandbox1.deckhouse-docs.flant.com'`, the actual URI would be:
   > `https://dex.sandbox1.deckhouse-docs.flant.com/callback`.
   >
   > To retrieve the Dex address (URI), run:
   >
   > ```
   > d8 k -n d8-user-authn get ingress dex -o jsonpath="{.spec.rules[*].host}"
   > ```

2. Create a DexProvider resource tailored to the specifics of your selected identity provider.
3. Enable the `user-authn` module (if it is currently disabled).

   This can be done either via the Deckhouse admin web interface or through the CLI.
   Below is an example using the Deckhouse CLI (requires kubectl context configured to access the cluster):

   Check the module status:

   ```
   d8 k get module user-authn
   ```

   Example output:

```
NAME         STAGE   SOURCE     PHASE       ENABLED   READY
user-authn           Embedded   Available   True      True
```

Enable the module via CL:

```
d8 system module enable user-authn
```

4. Configure the `user-authn` module.
   - Open the `user-authn` module settings (create a ModuleConfig resource named `user-authn` if it doesn't exist):

     ```
     d8 k edit mc user-authn
     ```

   - Specify the required module parameters in the `spec.settings` section. For more details about the `user-authn` module settings, refer to the module reference.

     Example `user-authn` configuration:

     ```
     apiVersion: deckhouse.io/v1alpha1
     kind: ModuleConfig
     metadata:
       name: user-authn
     spec:
       version: 3
       enabled: true
       settings:
         kubeconfigGenerator:
         - id: direct
           masterURI: https://159.89.5.247:6443
           description: "Direct access to kubernetes API"
     ```

   - To publish the Kubernetes API via an Ingress controller or Gateway API, enable the `apiserver.publishAPI.ingress.enabled` parameter in the `control-plane-manager` module settings:

     ```
     apiVersion: deckhouse.io/v1alpha1
     kind: ModuleConfig
     metadata:
       name: control-plane-manager
     spec:
       version: 3
       enabled: true
       settings:
         apiserver:
           publishAPI:
             ingress:
               enabled: true
     ```

   For more information on how to expose the Kubernetes API, see the `control-plane-manager` module documentation.

After setting up integration in the cluster with an external authentication provider, authentication through it will become possible in the platform's web interfaces. Information on setting up authentication for user applications can be found in the section Usage → IAM → Authentication.

Limiting the identities of a provider

A user who signs in through a DexProvider gets the email and the groups that the identity provider reports. The email is the Kubernetes username. As a result, users of the provider receive all permissions already granted in the cluster to those email addresses and groups.

To restrict the identities a provider can assert, use the `spec.allowedIdentities` parameter:

- `emails` and `emailDomains` describe the emails that the provider may assert. Dex refuses sign-in for any other email.
- `groups` describes the group names that the provider may assert. Dex keeps only the listed groups in the token and refuses sign-in if none is left.

The limits apply to every provider type, in addition to the group filters of the type, such as `oidc.allowedGroups`, `gitlab.groups` or `crowd.groups`. Group names are compared exactly. Nesting of Group objects does not extend the list.

Dex checks the limits at sign-in, at every refresh-token request and at Basic Authentication to the API server through an LDAP provider. Dex revokes refresh tokens for identities that no longer satisfy the limits, so narrowing the limits ends the existing sessions of the excluded users. For a Crowd or OIDC provider, basic-auth-proxy checks the credentials with the provider directly, so the limits do not apply to Basic Authentication.

The same limits define who can create or change the provider: you must be able to assign every role already granted to the emails and groups that the provider can report. Only a SuperAdmin can create or change a provider without an email limit. A Crowd or OIDC provider with `enableBasicAuth: true` counts as a provider without an email limit, so only a SuperAdmin can create it or turn on `enableBasicAuth` in it.

Example of a provider limited by an email domain and a group:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: contractors
spec:
  type: OIDC
  displayName: Contractors
  oidc:
    issuer: https://idp.contractor.example
    clientID: dex
    clientSecret: secret
  allowedIdentities:
    emailDomains: [contractor.example]
    groups: [contractors]
```

The following changes are not checked: changing `clientSecret` or `bindPW`, changing `displayName`, disabling the provider, narrowing the limits, reapplying the same manifest, and deleting the provider. Any other change is checked as connecting a new provider, including enabling a disabled provider.

If the check fails, the error lists the roles that the provider could receive, for example:

```
dexproviders.deckhouse.io "corp": the provider can assert identities that already carry roles [user-
authz:super-admin]; the requester's can-assign range is basic<=ClusterAdmin and does not cover them.
Narrow spec.allowedIdentities (emails, emailDomains, groups) or ask a SuperAdmin
```

Checking provider connectivity

The provider page of the DP web UI has the “Check connection” action. It creates a DexProviderCheck resource and waits until the `user-authn` controller writes the check result to its `status`.

The check verifies the following:

- the specified DexProvider exists and is enabled;
- the Dex discovery endpoint inside the cluster is reachable;
- the endpoint of the external provider is reachable.

For OIDC providers, the discovery document and JWKS are also read. For LDAP providers, TCP/TLS/StartTLS reachability is checked and, if Kerberos is enabled, the presence of the keytab Secret. The check also verifies that the provider accepts the client or service account credentials (GitHub, GitLab, Bitbucket Cloud, Crowd, OIDC, bind in LDAP) and warns about disabled TLS verification and certificates that expire soon. The check does not perform a full sign-in flow and does not check the password of a test user.

To run the check without the web UI, create a DexProviderCheck resource:

```
apiVersion: deckhouse.io/v1
kind: DexProviderCheck
metadata:
  name: check-github
spec:
  providerName: github
```

Then read its status:

```
d8 k get dexproviderchecks check-github -o yaml
```

OIDC (OpenID Connect) integration

Authentication via an OIDC provider requires registering a client (or creating an application). Follow your provider's documentation to do this (e.g., Okta, Keycloak, Gluu, or Blitz).

Specify the `clientID` and `clientSecret` obtained during setup in the DexProvider resource.

> **i** When registering an application with any OIDC provider, you must specify a redirect URI.
> For integration with DexProvider, use the following format: `https://dex.<publicDomainTemplate>/callback`, where `publicDomainTemplate` is the DNS name template of your cluster as defined in the `global` module.

> **i** To ensure proper token revocation on logout and to force re-authentication, set the `promptType` parameter to `login`.
> This guarantees that the user will be prompted to re-enter credentials during subsequent logins.

To configure fine-grained access control for users in applications:

- Add the `allowedUserGroups` parameter to the ModuleConfig of the target application.
- Assign the appropriate groups to the user, using identical group names on both the provider and Deckhouse sides.

Keycloak

During Keycloak configuration, select the appropriate `realm`, add a user in the Users section, and create a client in the Clients section with authentication enabled, which is required to generate a `clientSecret`. Then follow these steps:

1. In the Client scopes section, create a `scope` named `groups` and assign it a mapper `Group Membership` ("Client scopes" → "Client scope details" → "Mappers" → "Configure a new mapper"). Set values of "Name" and "Token Claim Name" as `groups` and turn off "Full group path".
2. In the previously created client, add this `scope` in the Client scopes tab ("Clients" → "Client details" → "Client Scopes" → "Add client scope").
3. In the "Valid redirect URIs", "Valid post logout redirect URIs", and "Web origins" fields in the client configuration, specify `https://dex.<publicDomainTemplate>/*`, where `publicDomainTemplate` is the cluster DNS name template defined in the `global` module.

Example provider configuration for Keycloak integration:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Use your realm name
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups
```

If email verification is not enabled in Keycloak, to properly use it as an identity provider, adjust the `Client Scopes` settings in one of the following ways:

- Delete the `Email verified` mapping (“Client Scopes” → “Email” → “Mappers”). This is required for proper processing of the `insecureSkipEmailVerified` field when it’s set to `true` and for correct permission assignment to users with unverified emails.
- If you can’t modify or delete the `Email verified` mapping, create a new Client Scope named `email_dkp` (or any other name) and add two mappings:
  - `email`: “Client Scopes” → `email_dkp` → “Add mapper” → “From predefined mappers” → `email`.
  - `email verified`: “Client Scopes” → `email_dkp` → “Add mapper” → “By configuration” → “Hardcoded claim”. Specify the following fields:
    - “Name”: `email verified`
    - “Token Claim Name”: `emailVerified`
    - “Claim value”: `true`
    - “Claim JSON Type”: `boolean`

  After that, in the client registered for the DP cluster in “Clients”, change `Client scopes` from `email` to `email_dkp`.

  In the DexProvider resource, specify `insecureSkipEmailVerified: true` and in the `.spec.oidc.scopes` field, change the Client Scope name to `email_dkp` following the example:

  ```
  scopes:
   - openid
   - profile
   - email_dkp
   - groups
  ```

⚠ With `insecureSkipEmailVerified: true`, Dex accepts unverified email addresses. Permissions are granted by email, so use this parameter only if the identity provider does not let users set an arbitrary email address.

Blitz Identity Provider

Example configuration of a provider for integration with Blitz Identity Provider:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim for obtaining user groups; groups are configured on
the Blitz Identity Provider side.
    clientID: clientID
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
    - profile
    - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC
```

Okta

Example configuration of a provider for integration with Okta:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientID: plainstring
    clientSecret: plainstring
    getUserInfo: true
```

After enabling integration with Okta, you can use Okta user groups to manage access rights. For example, you can specify a list of groups whose members are allowed to access Grafana.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access
```

SAML 2.0 integration

The example shows the provider’s settings for integration with a SAML 2.0 Identity Provider (e.g., AD FS, Okta, Keycloak).

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Corporate SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent
```

To configure the SAML Identity Provider:

1. Register Dex as a Service Provider (SP) in your IdP with the following settings:
   - **ACS URL (Assertion Consumer Service)**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **Entity ID**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **NameID format**: `persistent` or `emailAddress`
2. Configure attribute mappings in the IdP to send `email`, `name` (username), and `groups` attributes in the SAML assertion.
3. Export the IdP signing certificate and specify it in the `rootCAData` field of the DexProvider resource.

> **i** SAML does not support refresh tokens. Dex caches the user data from the first SAML assertion and returns it on subsequent refresh-token requests. The session lifetime is limited by the `refreshTokenAbsoluteLifetime` parameter of the `user-authn` module.

LDAP integration

To configure authentication, create a read-only account (service account) in your LDAP directory. This account will be used to perform search queries in the LDAP catalog. You can also configure user lockout after failed sign-in attempts for this type of integration. To do this, specify the provider type in the `passwordPolicy.lockout.applyToConnectors` parameter of the `user-authn` module.

In the DexProvider resource, specify the following parameters:

- `bindDN`: Full Distinguished Name (DN) of the created service account. For example: `cn=readonly,dc=example,dc=org`.
- `bindPW`: Password for the specified `bindDN`.

> **i** If your LDAP server allows anonymous access for search queries, the `bindDN` and `bindPW` parameters can be omitted.
> However, using authenticated access is recommended for improved security.
>
> The `bindPW` parameter must contain the password in plain text. Dex does not support hashed passwords in this field.

You can also enable support for:

- Basic Authentication for accessing the Kubernetes API using LDAP credentials. To do this, set the `enableBasicAuth: true` parameter in the DexProvider resource. For more information, see Access using Basic Authentication.
- Kerberos SSO (SPNEGO) for LDAP. To do this, enable the `spec.ldap.kerberos` block in the DexProvider resource, specify the keytab Secret in `keytabSecretName` and configure the other

parameters in it. For more details, see Kerberos SSO (SPNEGO) for LDAP.

Example configuration for integrating with Active Directory (Basic Authentication):

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----

    bindDN: cn=dex-readonly,cn=users,dc=example,dc=com
    bindPW: <PASSWORD>

    enableBasicAuth: true

    usernamePrompt: Email Address

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
      - userAttr: DN
        groupAttr: member
      nameAttr: cn
```

GitHub integration

You need to create a new application in your GitHub organization.

To do this, follow these steps:

1. Go to “Settings → Developer settings → OAuth Apps → New OAuth App”, and set the “Authorization callback URL” to `https://dex.<publicDomainTemplate>/callback`.
2. Use the generated `Client ID` and `Client Secret` in the DexProvider resource.

If the organization restricts third-party applications:

1. Go to “Settings → Applications → Authorized OAuth Apps → `<APPLICATION_NAME>`” and click “Send Request” to submit an approval request.
2. Ask an organization owner to approve the request.

Example configuration for integrating with GitHub:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company GitHub
  # Optional: disable the provider without deleting CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

GitLab integration

You need to create a new application in your GitLab project.

To do this, follow these steps:

1. For self-hosted GitLab: go to “Admin Area → Applications → New application” and set the “Redirect URI (Callback URL)” to `https://dex.<publicDomainTemplate>/callback`. Also, select the following scopes: `read_user`, `openid`.
2. For GitLab Cloud (gitlab.com): under the main account of the project, go to “User Settings → Applications → Add new application”, set the “Redirect URI (Callback URL)” to `https://dex.<publicDomainTemplate>/callback`, and select the **scopes**: `read_user`, `openid`.
3. Use the obtained `Application ID` and secret in the DexProvider resource.

> **i** For GitLab version 16 and above, enable the “Trusted” option when creating the application. This option is available under “Admin Area → Applications”. Marking the app as trusted allows skipping the authorization step for users, which can be useful in controlled environments.

Example configuration for integrating with GitLab:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated Gitlab
  gitlab:
    baseURL: https://gitlab.example.com
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Atlassian Crowd integration

In the relevant Atlassian Crowd project, you need to create a new Generic application.

You can also configure user lockout after failed sign-in attempts for this type of integration. To do this, specify the provider type in the `passwordPolicy.lockout.applyToConnectors` parameter of the `user-authn` module.

To do this, follow these steps:

1. Go to “Applications → Add application”.
2. Specify the obtained “Application Name” and “Password” in the DexProvider resource.
3. When specifying groups in the DexProvider resource, make sure their names are written in lowercase.
   This is necessary for correct group matching between Crowd and DP.

Example configuration for integrating with Atlassian Crowd:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseURL: https://crowd.example.com/crowd
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Basic Authentication to the Kubernetes API through a Crowd provider (`enableBasicAuth`) bypasses the identity limits of the provider, so only a SuperAdmin can enable it.

Bitbucket Cloud Integration

To configure authentication, you need to create a new OAuth consumer in the Bitbucket team menu.

Follow these steps:

1. Go to “Personal settings → Access management → OAuth consumers → Add consumer”, and specify `https://dex.<publicDomainTemplate>/callback` as the “Callback URL”.
2. Grant access:
   - “Account: Read” → allows retrieval of basic user information (e.g., email, username).
   - “Workspace membership: Read” → allows retrieval of user workspace membership information.
3. Use the obtained `Key` and secret in the DexProvider resource.

Example configuration for integrating with Bitbucket:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
    teams:
    - administrators
    - users
```

Additional resources

- Global parameters
- Module `control-plane-manager` documentation
- Module `user-authn` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Local authentication

In addition to external authentication providers, DP also supports local authentication.

Local authentication provides user verification and access management with support for configurable password policies, two-factor authentication (2FA), and group management. In the module parameters, you can configure password complexity, password history, password rotation, lockout after failed sign-in attempts and per-IP rate limiting of failed attempts. No external authentication system is required.

Local authentication involves creating User and Group resources in the cluster for static users and groups:

- A User object stores user information, including email and a hashed password (the password is not stored in plain text).
- A Group object defines a list of users grouped together.

Creating a static user

The recommended way to create a local user is the `d8 iam user create` command. It supports interactive password entry, automatic password generation, group assignment, and TTL for temporary users.

Examples:

Interactive password prompt (default when stdin is a terminal):

```
d8 iam user create anton --email anton@abc.com
```

Automatically generate a password (the generated password is shown in the command output once):

```
d8 iam user create anton --email anton@abc.com --generate-password
```

Read the password from stdin (for CI/CD pipelines):

```
echo "s3cret" | d8 iam user create anton --email anton@abc.com --password-stdin
```

Create the user and add them to groups (auto-creating groups if missing):

```
d8 iam user create anton --email anton@abc.com --generate-password --member-of admins --
create-groups
```

Create a temporary user with a TTL:

```
d8 iam user create anton --email anton@abc.com --generate-password --ttl 24h
```

As an alternative, you can create a User resource manually.

Example resource definition (note that the example includes a ttl):

```
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  password: $2a$10$etblbZ9yfZaKgbvysf1qguW3WULdMnxwWFrkoKpRH1yeWa5etjjAa
  ttl: 24h
```

Come up with a password and specify it in the `password` field as a raw bcrypt hash or a Base64-encoded bcrypt hash. To generate a Base64-encoded bcrypt hash, use the following command:

```
echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo
```

> **i** If the `htpasswd` command is not available, install the appropriate package:
>
> - `apache2-utils` — for Debian-based distributions.
> - `httpd-tools` — for CentOS-based distributions.
> - `apache2-htpasswd` — for ALT Linux.

> **⚠** You cannot create a User whose `spec.email` matches a `User` subject of an existing ClusterAuthorizationRule or AuthorizationRule: the new user would get the permissions of the rule unnoticed. If the match is intentional, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the User. For details, see How rules are applied.

Deleting a user

To delete a local user, use the `d8 iam user delete` command. By default, it also removes the user from all Group resources they belong to.

Examples:

Delete the user (and automatically remove them from all groups):

```
d8 iam user delete anton
```

Delete the user but keep their references in groups:

```
d8 iam user delete anton --keep-memberships
```

Local user operations

Password reset, forced password change, 2FA reset, user lock/unlock and logout operations are performed via the UserOperation resource. The `initiatorType` field indicates who initiated the operation: an administrator (`admin`), the system (`system`), or the user (`self`).

Administrative operations

Use the `d8 iam user` commands for administrative actions on local users. They create a UserOperation resource with `initiatorType: admin`, wait for the operation to complete, and print the result.

> **i** Deleting, recreating, or running a `ResetPassword`, `Reset2FA`, `Lock`, or `Unlock` operation on a user is allowed only if you already have permissions covering the roles that the user receives through their email or group membership, or you are explicitly allowed to assign those roles. The value of the `initiatorType` field does not affect this check: setting `initiatorType: self` does not bypass it. For details, see the `user-authn` module documentation. The same rule applies to DexProvider objects, see Limiting the identities of a provider.

The `ResetPassword`, `Reset2FA`, `Lock` and `Logout` operations end the sessions of the user at once, and the user has to authenticate again. For what ends and how fast, see UserOperation lifecycle and side effects. Only `Reset2FA` removes the user's 2FA secret.

After an administrator resets the password with `ResetPassword`, the user signs in with that password once. Dex then asks the user to set a new password that complies with the password policy.

Examples of using the `d8 iam user` commands:

- Interactive password reset:

  ```
  d8 iam user reset-password admin
  ```

- Reading the new password from stdin:

  ```
  echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
  ```

- Generating a new password automatically:

```
d8 iam user reset-password admin --generate-password
```

- Reset the password in hashed form (if the password is hashed, provide the bcrypt hash without Base64 encoding):

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

- 2FA reset:

```
d8 iam user reset2fa admin
```

- Locking a user for 30 minutes:

```
d8 iam user lock admin 30m
```

  The `d8 iam user lock` command locks a local user and accepts a duration in seconds, minutes or hours, for example `30s`, `10m` or `1h`. To lock a user for several days (`lock.for: 7d`) or permanently (`lock.for: permanent`), create a UserOperation as described in Creating UserOperation manually.

- User unlock:

```
d8 iam user unlock admin
```

By default, commands wait for the operation to complete. To only create a UserOperation and print its name, use the `--wait=false` flag.

Self-service operations

In the Deckhouse web interface, a user can sign out on all devices, and a local user can change their password. The web interface creates a UserOperation with `initiatorType: self` for the user who is signed in:

- Signing out on all devices creates a `Logout` operation. It ends every session of the user, including kubeconfig files.
- Changing the password creates a `RequirePasswordChange` operation. At the next sign-in, Dex asks for the current password and a new one that complies with the password policy. The operation ends no sessions.

Changing the password is available only for local accounts (the built-in `Local` connector). Users who sign in through external authentication providers change their password in the corresponding system.

For user-facing password change and reset scenarios, see Configuring authentication for applications.

Creating UserOperation manually

When running the `d8 iam user` CLI command is unavailable (e.g. in CI/CD, GitOps, or automation), you can create a UserOperation resource directly. Example — reset a local user's password (the `newPasswordHash` contains a bcrypt hash without Base64 encoding; the module encodes it automatically):

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: reset-password-admin
spec:
  user: admin
  type: ResetPassword
  initiatorType: admin
  resetPassword:
    newPasswordHash: "$2y$10$..."
```

For users of external providers of any type (LDAP, Atlassian Crowd, OIDC, SAML, GitHub, GitLab, Bitbucket Cloud), use `spec.target` instead of `spec.user`. The `Lock`, `Unlock` and `Logout` operations

are supported for external users. Example — lock an external user for 30 minutes:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: lock-external-user
spec:
  target:
    connectorID: my-ldap
    email: jane.doe@example.org
  type: Lock
  initiatorType: admin
  lock:
    for: "30m"
```

A lock applies only to sign-ins through the provider set in `connectorID`. It does not block a local user or a user of another provider with the same email. The operation fails if the user has never signed in through this provider.

A UserOperation is a **single-use** object: after creation, the module processes it and writes the result to `status.phase` (`Succeeded` or `Failed`). A UserOperation runs once, and changes to its spec after creation are ignored. Completed operations are **automatically deleted** after 24 hours.

⚠ The `ResetPassword`, `Reset2FA`, `Lock` and `Logout` operations end the sessions of the user at once, and the user has to authenticate again. For what ends and how fast, see UserOperation lifecycle and side effects. `ResetPassword`, `Lock` and `Logout` keep the user's 2FA secret.

Viewing account state

The `user-authn` module keeps a UserAccount object for each local user and for each user who has signed in through an external provider. It shows whether the account is locked, until when, and how many failed sign-in attempts were counted. Do not create or edit UserAccount objects.

To list the accounts, run:

```
d8 k get useraccounts
```

Adding a user to a group

⚠ It is forbidden to use users and groups with the `system:` prefix.
Authentication attempts by such users or members of such groups will be rejected, and a corresponding warning will appear in the `kube-apiserver` logs.

⚠ You cannot create a Group whose `spec.name` matches a `Group` subject of an existing ClusterAuthorizationRule or AuthorizationRule: the members of the new group would get the permissions of the rule unnoticed. If the match is intentional, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the Group. For details, see How rules are applied.

The recommended way to manage groups is the `d8 iam group` command. Examples:

Create a group:

```
d8 iam group create admins
```

Add a user to a group:

```
d8 iam group add-member admins user anton
```

Remove a user from a group:

```
d8 iam group remove-member admins user anton
```

Delete a group:

```
d8 iam group delete admins
```

As an alternative, you can create a Group resource manually.

Example resource definition:

```
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin
```

Where `members` is a list of users belonging to the group.

Once the group is created and includes all necessary users, proceed by configuring authorization.

Configuring password policy

Password policy allows controlling password complexity, rotation, and automatic lockout after consecutive failed login attempts.

An administrator can lock or unlock a user with `d8 iam user lock` / `unlock` (or a UserOperation) without enabling `passwordPolicy.lockout`. The `lockout` section only turns on automatic lockout after the configured number of failed attempts.

Dex stores an automatic lock itself and does not create a UserOperation for it. Local users are always covered. LDAP and Atlassian Crowd users are locked only if their provider type is listed in `passwordPolicy.lockout.applyToConnectors`. To limit failed sign-in attempts per client IP address, use the `rateLimit` parameter.

To set up a password policy, use the `passwordPolicy` field in the configuration of the `user-authn` module.

Examples of policies:

- Without custom complexity rules
- With custom complexity rules

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Custom
      custom:
        minLength: 10
        specialCharacters: true
        numbers: false
        capitalized: true
        repeatedChars: false
      passwordHistoryLimit: 10
```

Configuring two-factor authentication (2FA)

2FA increases security by requiring a code from a TOTP authenticator application (for example, Google Authenticator) during login.

To set up 2FA, use the `twoFactor` field in the configuration of the `user-authn` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 4
  enabled: true
  settings:
    twoFactor:
      enabled: true
      issuerName: "awesome-app"
```

Field description:

- `enabled`: Enables or disables 2FA for all static users.
- `issuerName`: Name displayed in the authenticator application when adding an account.

> **i** After enabling 2FA, each user must register in the authenticator application during their first login.

Additional resources

- Module `user-authn` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Accessing the Kubernetes API via load balancer

DP allows using authentication when accessing the Kubernetes API. In this case, a user can generate a `kubectl` configuration via the DP web UI to securely access the Kubernetes API through a traffic balancer (Ingress controller).

To configure access, follow these steps:

1. Enable Kubernetes API publishing. To do this, set the parameter `apiserver.publishAPI.ingress.enabled: true` in the `control-plane-manager` module settings or via the Deckhouse Platform web interface.

   Example module configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     enabled: true
     version: 3
     settings:
       apiserver:
         publishAPI:
           ingress:
             enabled: true
   ```

2. Open the DP web UI.
   The published API becomes available for kubeconfig generation automatically after enabling API publishing (the `apiserver.publishAPI.ingress.enabled` parameter) in the `control-plane-manager` module.

3. Generate the `kubectl` configuration.
   The web UI provides a set of commands to configure `kubectl`.
   These commands can be copied and pasted into the terminal.
   Authentication will be performed using an OIDC token issued by Dex.
   If the provider supports session renewal, the configuration will include a `refresh token`, allowing access to be extended without re-authentication.

4. Configure additional API access points.
   In the `user-authn` module configuration, you can specify additional connection points (kube-apiserver), each with its own description and CA certificates.
   This may be useful if the cluster is accessible through different networks — for example, via VPN or public IP:

   ```
   settings:
     kubeconfigGenerator:
     - id: direct
       masterURI: https://159.89.5.247:6443
       description: "Direct access to kubernetes API"
     - id: second
       masterURI: https://5.6.7.8:6443
       description: "Second access to kubernetes API"
   ```

How API access protection works in Kubernetes

In Deckhouse Platform, you can safely expose the Kubernetes API externally using an Ingress controller or Gateway API while maintaining access control (for more information on how to expose the Kubernetes API, see the `control-plane-manager` module documentation). API exposure is configured via the `control-plane-manager` module, while authentication is handled via the `user-authn` module. You can configure:

- A list of trusted IP addresses or networks allowed to access the API.
- The Ingress controller through which access will be provided.

To configure:

1. Enable API publishing as shown in the example above.
2. Configure access restrictions. In the `control-plane-manager` module configuration, you can specify:
   - A list of IP addresses or networks allowed to access (`apiserver.publishAPI.ingress.whitelistSourceRanges`).
   - The Ingress controller to be used for publishing (`apiserver.publishAPI.ingress.ingressClass`).

3. Use the DP web UI.
   Users will be able to securely access the API using the kubeconfig generated there.
   This kubeconfig will include the OIDC token and the Ingress connection settings.

The following will be automatically configured when API publishing is enabled:

- A CA certificate will be generated and added to the kubeconfig.
- Login via Dex with OIDC support will be configured.

The JWT authenticator for Dex does not depend on API publishing. It is enabled together with the `user-authn` module: using the `control-plane-manager` module, DP passes kube-apiserver an AuthenticationConfiguration with a JWT authenticator for Dex.

Access using Basic Authentication (LDAP)

In addition to OIDC, you can configure direct access to the API using Basic Authentication (username and password). In this case, credentials are verified against LDAP-compatible directory service.

To configure:

1. Enable API publishing (the `apiserver.publishAPI` parameter of the `control-plane-manager` module).
2. Configure an LDAP provider in the `user-authn` module and enable the `enableBasicAuth: true` option.

> ⚠ Only one provider in the cluster can have `enableBasicAuth` enabled.

Basic Authentication is also available for Crowd and OIDC providers (the `enableBasicAuth` parameter in the `crowd` or `oidc` section). Through an LDAP provider, Basic Authentication goes through Dex and keeps the identity limits of the provider. Through a Crowd or OIDC provider, it bypasses these limits, so only a SuperAdmin can enable it.

After this, users can configure their `kubeconfig` by specifying their LDAP username and password:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Path to CA certificate or insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default
```

Kerberos (SPNEGO) SSO for LDAP

Dex supports passwordless Kerberos (SPNEGO) flow for the LDAP connector. The mechanism works as follows:

1. The browser, trusting the Dex host, sends `Authorization: Negotiate …`.
2. Dex validates the Kerberos ticket against the keytab and skips the login/password input form.
3. Dex matches the principal with the LDAP name, retrieves the groups, and completes the OIDC flow.

> **i** To configure Kerberos (SPNEGO) SSO, LDAP must have an account with read-only privileges (service account). For more information on configuration, see the section LDAP Integration.

Enabling Kerberos (SPNEGO) SSO for LDAP:

1. In AD/KDC, create/provision an SPN `HTTP/<DEX_FQDN>` for a service account and generate a keytab.
2. In the cluster, create a Secret in the `d8-user-authn` namespace with the `krb5.keytab` data key.
3. In the LDAP DexProvider resource, enable `spec.ldap.kerberos`:
   - `enabled: true`
   - `keytabSecretName: <secret name>`
   - optional: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`

Dex will mount the keytab automatically and start accepting SPNEGO. A server-side `krb5.conf` is not required — tickets are validated using the keytab.

Example of configuring SSO via Kerberos (SPNEGO) for LDAP (extension of the LDAP provider specification):

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=dex-readonly,cn=users,dc=example,dc=com
    bindPW: <PASSWORD>
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
      - userAttr: uid
        groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab  # Secret in d8-user-authn with key
'krb5.keytab'.
      expectedRealm: EXAMPLE.COM           # Optional, case-insensitive match.
      usernameFromPrincipal: SAMAccountName  #
Localpart|SAMAccountName|UserPrincipalName.
      fallbackToPassword: false            # The default is false; if true, a
login/password form is rendered when the `Authorization: Negotiate` header is
missing or invalid.
```

Notes:

- The Secret `dex-kerberos-keytab` must exist in the `d8-user-authn` namespace and have a data key named exactly `krb5.keytab`.
- A single Dex Pod can serve multiple LDAP+Kerberos providers. Each provider mounts its own keytab; a shared `krb5.conf` is not required (Dex validates tickets offline using the keytab).
- A DexProvider with `enabled: true` in `spec.ldap.kerberos` must have `keytabSecretName` set, otherwise the object is rejected when saved. To turn Kerberos off and keep its settings, set `enabled: false`.

Additional resources

- Module `control-plane-manager` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Resource configuration

Deckhouse Platform allows you to configure resource limits and requests for all components. By default, the following values are used:

- **Dex OIDC Provider** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Kubeconfig Generator** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Basic Auth Proxy** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Dex Authenticator** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Redis** (used by Dex Authenticator) — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)

> When Vertical Pod Autoscaler (VPA) is enabled, the resource limits are managed automatically by VPA, but you can still configure the minimum and maximum allowed values through the `resources` section.

Example configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: v1
  settings:
    resources:
      dex:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      kubeconfigGenerator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      basicAuthProxy:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      dexAuthenticator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      redis:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
```

Additional resources

- Module `vertical-pod-autoscaler` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Authorization

In Deckhouse Platform (DP), authorization is based on the standard Kubernetes Role-Based Access Control (RBAC) mechanism. This allows for flexible access control for different users, groups, and service accounts, ensuring security and operational control in the cluster.

DP supports two role models:

- Granular (recommended). Access permissions are configured the standard Kubernetes RBAC way: by creating RoleBinding or ClusterRoleBinding resources — or, for project-wide access, the ProjectRoleBinding and ClusterProjectRoleBinding resources of the `multitenancy-manager` module.
- Basic. The end-to-end authorization subsystem extends the standard RBAC mechanism with custom resources — ClusterAuthorizationRule and AuthorizationRule. Support for this model will be discontinued in future releases.

Both models are supported by the `user-authz` module and can be used at the same time — the permissions they grant are summed up (for details, see the `user-authz` module FAQ). The choice of model depends on security requirements and usage scenarios.

Combined use of ClusterAuthorizationRule, AuthorizationRule, and RBAC

If multitenancy mode is enabled in the cluster (`enableMultiTenancy: true`), a user's effective permissions are the union of the permissions from all of the following sources:

- **permissions from ClusterAuthorizationRule** — apply only within the namespaces allowed by the `namespaceSelector` parameter (or the deprecated `limitNamespaces` parameter);
- **permissions from AuthorizationRule** — apply within the namespace where the AuthorizationRule resource is created;
- **permissions from plain RoleBinding resources** — apply within the namespace where the RoleBinding resource is created;
- **permissions from plain ClusterRoleBinding resources** not created by the `user-authz` module — apply cluster-wide.

The namespace restrictions set in a ClusterAuthorizationRule apply only to the permissions granted by that ClusterAuthorizationRule. They do not override permissions granted via a RoleBinding, ClusterRoleBinding, or AuthorizationRule in other namespaces. At the same time, the permissions granted by the ClusterAuthorizationRule do not extend to those other namespaces.

The namespace restrictions are enforced by the authorization webhook, so they do not apply to the identities that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. A ClusterAuthorizationRule that names such a ServiceAccount, directly or through a group such as `system:serviceaccounts`, grants its access level in every namespace. Create the ServiceAccounts of external systems in ordinary namespaces.

For example, if a user has a ClusterAuthorizationRule with `accessLevel: Editor` restricted to the `ns-a` namespace, as well as a RoleBinding with the `view` role in the `ns-b` namespace, they get `Editor`-level permissions in `ns-a` and read-only permissions in `ns-b`. The permissions granted via the RoleBinding are not restricted by the ClusterAuthorizationRule, and the `Editor` access level set by the ClusterAuthorizationRule does not extend to the `ns-b` namespace.

Who gets access and when

There are two scenarios for granting access in DP:

- Granting access to users for working via command-line clients, web UI, and other tools used for administration, development, and cluster management.
- Granting access to service accounts for automating tasks such as application deployment and updates (most commonly using the IaC approach). Examples of such services include CI/CD systems, monitoring systems, and others.

After successful authentication, users and service accounts are granted access to cluster resources based on the configured authorization settings.

User authentication

DP supports multiple user authentication methods. For details, refer to User authentication.

Service account authentication

In Kubernetes, service accounts (ServiceAccount) are special accounts used to automate tasks and interact with the cluster API. They enable applications and services to securely communicate with the Kubernetes API.

⚠ Do not create ServiceAccounts for external systems in `kube-system` or in namespaces whose names start with `d8-`. The authorization webhook does not check the ServiceAccounts of these namespaces, so the namespace limits of a ClusterAuthorizationRule (`namespaceSelector`, `limitNamespaces`, `allowAccessToSystemNamespaces`) do not apply to them. Their access level applies in every namespace of the cluster, including system namespaces.

You may need to create a ServiceAccount with access to the Kubernetes API, for example, when setting up application deployment through a CI system. To create a ServiceAccount and grant it access, follow these steps:

i The example uses the basic role model.

1. Create a namespace for CI and a ServiceAccount in it:

```
d8 k create -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: ci
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: ci
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: ci
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

2. Grant the ServiceAccount the necessary permissions using the ClusterAuthorizationRule custom resource:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-deploy
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci
  accessLevel: Editor
EOF
```

   If multitenancy mode is enabled (the `enableMultiTenancy` parameter of the `user-authz` module), limit the namespaces available to the ServiceAccount with the `namespaceSelector` parameter.

3. Set the variable values (they are used later) by running the following commands (**insert your own values**):

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy.my-cluster
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config
```

4. Generate the `cluster` section in the kubectl configuration file:

   Use one of the following options to access the cluster API server:

   - If there is direct access to the API server:
     1. Get the Kubernetes cluster CA certificate:

        ```
        d8 k get cm kube-root-ca.crt -o jsonpath='{ .data.ca\.crt }' > /tmp/ca.crt
        ```

     2. Generate the `cluster` section (the API server IP address is used for access):

        ```
        d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
          --server=https://$(d8 k get endpointslices -l kubernetes.io/service-name=kubernetes
        -o json | jq -rc '.items[0] | "\(.endpoints[0].addresses[0]):\
        (.ports[0].port)"') \
          --certificate-authority=/tmp/ca.crt \
          --kubeconfig=$FILE_NAME
        ```

   - If there is no direct access to the API server, use one of the following options:
     - enable access to the API server through the Ingress controller (the apiserver.publishAPI parameter of the `control-plane-manager` module) and specify the addresses requests will come from (the whitelistSourceRanges parameter);
     - specify the addresses requests will come from in a separate Ingress controller (the acceptRequestsFrom parameter).
   - If a non-public CA is used:
     1. Get the CA certificate from the Secret of the `kubernetes-api` Ingress in the `kube-system` namespace. This is the certificate of the `api.%s` domain:

        ```
        d8 k -n kube-system get secrets -o json \
          $(d8 k -n kube-system get ing kubernetes-api -o jsonpath="
        {.spec.tls[0].secretName}") \
          | jq -rc '.data."ca.crt" // .data."tls.crt"' \
          | base64 -d > /tmp/ca.crt
        ```

     2. Generate the `cluster` section (an external domain and the CA are used for access):

        ```
        d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
          --server=https://$(d8 k -n kube-system get ing kubernetes-api -ojson | jq
        '.spec.rules[].host' -r) \
          --certificate-authority=/tmp/ca.crt \
          --kubeconfig=$FILE_NAME
        ```

   - If a public CA is used, generate the `cluster` section (an external domain is used for access):

     ```
     d8 k config set-cluster $CLUSTER_NAME \
       --server=https://$(d8 k -n kube-system get ing kubernetes-api -ojson | jq
     '.spec.rules[].host' -r) \
       --kubeconfig=$FILE_NAME
     ```

   - If the API server is published via Gateway API, the cluster has the `kubernetes-api` HTTPRoute in the `kube-system` namespace instead of the Ingress. In the commands above, use the host from it instead of `.spec.rules[].host`:

     ```
     d8 k -n kube-system get httproute kubernetes-api -o jsonpath='{.spec.hostnames[0]}'
     ```

     Take the name of the Secret with the CA certificate from the `control-plane-manager`

ListenerSet instead of `.spec.tls[0].secretName`:

```
d8 k -n kube-system get listenerset control-plane-manager -o
jsonpath='{.spec.listeners[?(@.name=="api")].tls.certificateRefs[0].name}'
```

5. Generate the `user` section with the token from the ServiceAccount Secret in the kubectl configuration file:

```
d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n ci get secret gitlab-runner-deploy-token -o json |jq -r
'.data["token"]' | base64 -d) \
  --kubeconfig=$FILE_NAME
```

6. Generate the context in the kubectl configuration file:

```
d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME
```

7. Set the generated context as the default one in the kubectl configuration file:

```
d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

Additional resources

- Module `control-plane-manager` parameters
- Module `ingress-nginx` custom resources
- Module `multitenancy-manager` custom resources
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Basic authorization model

To use the basic role model, the `user-authz` module must be enabled in the cluster. This module creates a set of cluster roles (ClusterRole) suitable for most user and group access management tasks.

> The module implements two role-based models: the granular one (recommended) and the basic one (this page), built around the ClusterAuthorizationRule and AuthorizationRule resources — its support will be discontinued in future releases.
>
> The models are not resource-compatible — automatic conversion is impossible — but they can be used at the same time: the permissions of both models are summed up.

Key features of the basic role model:

- Implements a role-based end-to-end authorization subsystem that extends the standard RBAC functionality.
- Access is configured with the custom resources ClusterAuthorizationRule and AuthorizationRule, on top of Kubernetes RBAC.
- Controls access to scaling tools (via the `allowScale` parameter in the ClusterAuthorizationRule or AuthorizationRule resource).
- Controls access to port forwarding (via the `portForwarding` parameter in the ClusterAuthorizationRule or AuthorizationRule resource).
- Manages the list of allowed namespaces using the labelSelector format (via the `namespaceSelector` parameter in the ClusterAuthorizationRule resource).
- Namespace restrictions (multitenancy mode, `namespaceSelector`, `limitNamespaces`) are available in the DP Ultimate/EE, Certified Lite, Certified Pro, BE, SE, SE+, and CSE editions.

High-level roles used in the basic model

In addition to RBAC, the basic role model implemented via the `user-authz` module provides a convenient set of high-level roles:

Role	Examples of allowed actions	Limitations
User	View Pods, logs, and Deployments	Can’t access secrets, ports, or containers
PrivilegedUser	Access containers (via `kubectl exec`), view secrets, remove Pods	Can’t modify a Deployment or Service
Editor	Create and remove a Deployment, Service, or ConfigMap	Can’t access a ReplicaSet or ClusterRoles
Admin	Create and modify Pods, create ServiceAccount tokens (`serviceaccounts/token`), manage Role, RoleBinding, and AuthorizationRule objects, delete auxiliary objects (for example, ReplicaSet and the challenges and orders of the `acme.cert-manager.io` group)	Can’t access cluster-wide resources
ClusterEditor	Manage a limited set of cluster-wide objects used in application tasks: DaemonSet, StorageClass, CustomResourceDefinition, GatewayClass, and the objects that modules add at this level (for example, KeepalivedInstance)	Can’t manage namespaces, quotas, ClusterRole (read-only access), and ClusterRoleBinding
ClusterAdmin	Manage cluster-wide service objects: namespaces, quotas, NetworkPolicy, the proxy subresources of nodes, ClusterAuthorizationRule, ClusterRole, and ClusterRoleBinding	Can create ClusterAuthorizationRule objects only up to the `ClusterAdmin` level, and ClusterRoleBinding objects only to roles whose permissions it already holds
SuperAdmin	Any actions are allowed (including `*` in RBAC), but with consideration to `limitNamespaces`	Limitations can be applied only using cluster policies

⚠ Multitenancy mode restricts access to namespaced resources only. Subjects with the `ClusterAdmin` or `SuperAdmin` access level can still create cluster-wide bindings. The authorization webhook does not check control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`, so the namespace restrictions do not apply to them.

The `limitNamespaces` and `allowAccessToSystemNamespaces` parameters of the ClusterAuthorizationRule resource are deprecated. To restrict a rule to certain namespaces, use `namespaceSelector`. To grant access to all namespaces, including the system ones, use `namespaceSelector.matchAny: true`. If `namespaceSelector` is set, the deprecated parameters are ignored.

In multitenancy mode, if the authorization webhook becomes unavailable, requests are **denied**, not allowed: the webhook runs before RBAC and is configured with `failurePolicy: Deny`, so a request it cannot answer does not reach RBAC. The exception is the identities listed in `matchConditions` of the AuthorizationConfiguration: control plane components, `kubernetes-api-proxy`, kubelets, the ServiceAccounts of the `kube-system` and `d8-*` namespaces, and anonymous kube-apiserver health probes (`/livez`, `/readyz`, `/healthz`) bypass the webhook, so the cluster keeps running and Deckhouse can restore the webhook.

Some platform components place their objects into user namespaces (for example, Dex authenticator

pods) and label them `heritage: deckhouse`. No access level, `SuperAdmin` included, allows creating, modifying and deleting such objects or running `exec`, `attach` and `port-forward` in such pods. For details, see Admin level restrictions and superadmin rights.

Default access list for each high-level role

> **i** This list is a static snapshot. The actual set of rules also depends on which DP modules are enabled in the cluster.

Verb shortcuts:

- read: `get`, `list`, `watch`
- read-write: `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, `update`
- write: `create`, `delete`, `deletecollection`, `patch`, `update`

Role `User`:

```
read:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - apps/deployments
    - apps/replicasets
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - cert-manager.io/certificaterequests
    - cert-manager.io/certificates
    - cert-manager.io/clusterissuers
    - cert-manager.io/issuers
    - cilium.io/ciliumclusterwidenetworkpolicies
    - cilium.io/ciliumnetworkpolicies
    - config.gatekeeper.sh/configs
    - configmaps
    - connection.gatekeeper.sh/connections
    - constraints.gatekeeper.sh/*
    - deckhouse.io/applicationpackages
    - deckhouse.io/applicationpackageversions
    - deckhouse.io/applications
    - deckhouse.io/awsinstanceclasses
    - deckhouse.io/azureinstanceclasses
    - deckhouse.io/clusterprojectrolebindings
    - deckhouse.io/deschedulers
    - deckhouse.io/dexauthenticators
    - deckhouse.io/dexclients
    - deckhouse.io/dvpinstanceclasses
    - deckhouse.io/dynamixinstanceclasses
    - deckhouse.io/gcpinstanceclasses
    - deckhouse.io/huaweicloudinstanceclasses
    - deckhouse.io/hubblemonitoringconfigs
    - deckhouse.io/instances
    - deckhouse.io/keepalivedinstances
    - deckhouse.io/localpathprovisioners
    - deckhouse.io/nodeextensionrequests
    - deckhouse.io/nodegroups
    - deckhouse.io/nodeoperations
    - deckhouse.io/openstackinstanceclasses
    - deckhouse.io/operationpolicies
    - deckhouse.io/projectnamespaces
    - deckhouse.io/projectrolebindings
    - deckhouse.io/projecttemplates
    - deckhouse.io/securitypolicies
    - deckhouse.io/securitypolicyexceptions
    - deckhouse.io/vcdaffinityrules
    - deckhouse.io/vcdinstanceclasses
    - deckhouse.io/vsphereinstanceclasses
    - deckhouse.io/yandexinstanceclasses
    - deckhouse.io/zvirtinstanceclasses
    - discovery.k8s.io/endpointslices
    - endpoints
    - events
    - events.k8s.io/events
```

```
    - expansion.gatekeeper.sh/expansiontemplate
    - extensions.istio.io/wasmplugins
    - extensions/daemonsets
    - extensions/deployments
    - extensions/ingresses
    - extensions/replicasets
    - extensions/replicationcontrollers
    - externaldata.gatekeeper.sh/providers
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gatewayclasses
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - infrastructure.cluster.x-k8s.io/deckhouseclusters
    - infrastructure.cluster.x-k8s.io/deckhousemachines
    - infrastructure.cluster.x-k8s.io/deckhousemachinetemplates
    - infrastructure.cluster.x-k8s.io/dynamixclusters
    - infrastructure.cluster.x-k8s.io/dynamixmachines
    - infrastructure.cluster.x-k8s.io/dynamixmachinetemplates
    - infrastructure.cluster.x-k8s.io/huaweicloudclusters
    - infrastructure.cluster.x-k8s.io/huaweicloudmachines
    - infrastructure.cluster.x-k8s.io/huaweicloudmachinetemplates
    - infrastructure.cluster.x-k8s.io/vcdclusters
    - infrastructure.cluster.x-k8s.io/vcdclustertemplates
    - infrastructure.cluster.x-k8s.io/vcdmachines
    - infrastructure.cluster.x-k8s.io/vcdmachinetemplates
    - infrastructure.cluster.x-k8s.io/zvirtclusters
    - infrastructure.cluster.x-k8s.io/zvirtmachines
    - infrastructure.cluster.x-k8s.io/zvirtmachinetemplates
    - limitranges
    - metrics.k8s.io/nodes
    - metrics.k8s.io/pods
    - multitenancy.deckhouse.io/availableclusterresources
    - mutations.gatekeeper.sh/assign
    - mutations.gatekeeper.sh/assignimage
    - mutations.gatekeeper.sh/assignmetadata
    - mutations.gatekeeper.sh/modifyset
    - namespaces
    - network.deckhouse.io/egressgatewaypolicies
    - network.deckhouse.io/egressgateways
    - network.deckhouse.io/metalloadbalancerbgppeers
    - network.deckhouse.io/metalloadbalancerclasses
    - network.deckhouse.io/metalloadbalancerconfigurations
    - network.deckhouse.io/metalloadbalancerpools
    - network.deckhouse.io/servicewithhealthchecks
    - network.deckhouse.io/waypointinstances
    - networking.istio.io/destinationrules
    - networking.istio.io/envoyfilters
    - networking.istio.io/gateways
    - networking.istio.io/serviceentries
    - networking.istio.io/sidecars
    - networking.istio.io/virtualservices
    - networking.istio.io/workloadentries
    - networking.istio.io/workloadgroups
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - nodes
    - persistentvolumeclaims
    - persistentvolumes
    - pods
    - pods/log
    - policy/poddisruptionbudgets
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - replicationcontrollers
    - resourcequotas
    - security.istio.io/authorizationpolicies
    - security.istio.io/peerauthentications
    - security.istio.io/requestauthentications
    - serviceaccounts
    - services
    - status.gatekeeper.sh/configpodstatuses
    - status.gatekeeper.sh/connectionpodstatuses
    - status.gatekeeper.sh/constraintpodstatuses
    - status.gatekeeper.sh/constrainttemplatepodstatuses
    - status.gatekeeper.sh/expansiontemplatepodstatuses
```

```
    - status.gatekeeper.sh/mutatorpodstatuses
    - status.gatekeeper.sh/providerpodstatuses
    - storage.k8s.io/storageclasses
    - syncset.gatekeeper.sh/syncsets
    - telemetry.istio.io/telemetries
    - templates.gatekeeper.sh/constrainttemplates
```

Role `PrivilegedUser` (includes all rules from the role `User`):

```
create:
    - pods/eviction
create,get:
    - pods/attach
    - pods/exec
delete,deletecollection:
    - pods
read:
    - secrets
```

Role `Editor` (includes all rules from the role `User`, `PrivilegedUser`):

```
get,patch:
    - pods/resize
write:
    - apps/deployments
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - cert-manager.io/certificates
    - configmaps
    - deckhouse.io/dexauthenticators
    - deckhouse.io/dexclients
    - discovery.k8s.io/endpointslices
    - endpoints
    - extensions/deployments
    - extensions/ingresses
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - network.deckhouse.io/servicewithhealthchecks
    - network.deckhouse.io/waypointinstances
    - networking.istio.io/destinationrules
    - networking.istio.io/gateways
    - networking.istio.io/serviceentries
    - networking.istio.io/sidecars
    - networking.istio.io/virtualservices
    - networking.istio.io/workloadentries
    - networking.istio.io/workloadgroups
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - persistentvolumeclaims
    - policy/poddisruptionbudgets
    - secrets
    - security.istio.io/authorizationpolicies
    - security.istio.io/peerauthentications
    - security.istio.io/requestauthentications
    - serviceaccounts
    - services
```

Role `Admin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create:
    - serviceaccounts/token
create,patch,update:
    - pods
delete,deletecollection:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - apps/replicasets
    - cert-manager.io/certificaterequests
    - extensions/replicasets
read-write:
    - deckhouse.io/authorizationrules
write:
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - cert-manager.io/issuers
    - deckhouse.io/applications
    - extensions.istio.io/wasmplugins
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - telemetry.istio.io/telemetries
```

Role `ClusterEditor` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create,delete,patch:
    - deckhouse.io/nodeoperations
delete,deletecollection:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - cert-manager.io/certificaterequests
get,list:
    - templates.internal.deckhouse.io/nodeconfigtemplates
patch,update:
    - nodes
read:
    - deckhouse.io/containerdintegritypolicies
    - deckhouse.io/ingressistiocontrollers
    - deckhouse.io/istiofederations
    - deckhouse.io/istiomulticlusters
    - multitenancy.deckhouse.io/grantableclusterresourcedefinitions
    - multitenancy.deckhouse.io/grantableclusterresourcereferences
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - sailoperator.io/istiocnis
    - sailoperator.io/istiorevisions
    - sailoperator.io/istiorevisiontags
    - sailoperator.io/istios
    - sailoperator.io/ztunnels
read-write:
    - deckhouse.io/nodegroupconfigurations
    - deckhouse.io/staticinstances
    - multitenancy.deckhouse.io/clusterresourcegrantpolicies
write:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - cert-manager.io/clusterissuers
    - cert-manager.io/issuers
    - deckhouse.io/applications
    - deckhouse.io/hubblemonitoringconfigs
    - deckhouse.io/instances
    - deckhouse.io/keepalivedinstances
    - deckhouse.io/nodeextensionrequests
    - deckhouse.io/nodegroups
    - extensions.istio.io/wasmplugins
    - extensions/daemonsets
    - gateway.networking.k8s.io/gatewayclasses
    - network.deckhouse.io/egressgatewaypolicies
    - network.deckhouse.io/egressgateways
    - storage.k8s.io/storageclasses
    - telemetry.istio.io/telemetries
```

Role `ClusterAdmin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`):

```
create:
    - authorization.deckhouse.io/bulksubjectaccessreviews
    - authorization.deckhouse.io/roleaccessreports
    - authorization.deckhouse.io/subjectaccessreports
```

```
    - deckhouse.io/dexauthenticators/allow-access-to-kubernetes
    - deckhouse.io/dexclients/allow-access-to-kubernetes
get,list,patch,update,watch:
    - control-plane.deckhouse.io/controlplanenodes
patch,update:
    - deckhouse.io/vcdaffinityrules
    - infrastructure.cluster.x-k8s.io/deckhouseclusters
    - infrastructure.cluster.x-k8s.io/deckhousemachines
    - infrastructure.cluster.x-k8s.io/deckhousemachinetemplates
    - infrastructure.cluster.x-k8s.io/dynamixclusters
    - infrastructure.cluster.x-k8s.io/dynamixmachines
    - infrastructure.cluster.x-k8s.io/dynamixmachinetemplates
    - infrastructure.cluster.x-k8s.io/huaweicloudclusters
    - infrastructure.cluster.x-k8s.io/huaweicloudmachines
    - infrastructure.cluster.x-k8s.io/huaweicloudmachinetemplates
    - infrastructure.cluster.x-k8s.io/vcdclusters
    - infrastructure.cluster.x-k8s.io/vcdclustertemplates
    - infrastructure.cluster.x-k8s.io/vcdmachines
    - infrastructure.cluster.x-k8s.io/vcdmachinetemplates
    - infrastructure.cluster.x-k8s.io/zvirtclusters
    - infrastructure.cluster.x-k8s.io/zvirtmachines
    - infrastructure.cluster.x-k8s.io/zvirtmachinetemplates
proxy:
    - nodes
read:
    - control-plane.deckhouse.io/controlplaneoperations
    - deckhouse.io/nodestaticpodrequests
    - nfd.k8s-sigs.io/nodefeaturegroups
    - nfd.k8s-sigs.io/nodefeaturerules
    - nfd.k8s-sigs.io/nodefeatures
read-write:
    - deckhouse.io/clusterauthorizationrules
    - deckhouse.io/deckhousereleases
    - deckhouse.io/dexproviderchecks
    - deckhouse.io/dexproviders
    - deckhouse.io/groups
    - deckhouse.io/moduleconfigs
    - deckhouse.io/modulepulloverrides
    - deckhouse.io/modulereleases
    - deckhouse.io/modules
    - deckhouse.io/modulesources
    - deckhouse.io/moduleupdatepolicies
    - deckhouse.io/nodeusers
    - deckhouse.io/packagerepositories
    - deckhouse.io/packagerepositoryoperations
    - deckhouse.io/sshcredentials
    - deckhouse.io/useraccounts
    - deckhouse.io/useroperations
    - deckhouse.io/users
    - nodes/configz
    - nodes/healthz
    - nodes/log
    - nodes/metrics
    - nodes/pods
    - nodes/proxy
    - nodes/stats
update:
    - namespaces/finalize
write:
    - cilium.io/ciliumclusterwidenetworkpolicies
    - cilium.io/ciliumnetworkpolicies
    - config.gatekeeper.sh/configs
    - connection.gatekeeper.sh/connections
    - constraints.gatekeeper.sh/*
    - deckhouse.io/applicationpackages
    - deckhouse.io/applicationpackageversions
    - deckhouse.io/awsinstanceclasses
    - deckhouse.io/azureinstanceclasses
    - deckhouse.io/clusterprojectrolebindings
    - deckhouse.io/containerdintegritypolicies
    - deckhouse.io/deschedulers
    - deckhouse.io/dvpinstanceclasses
    - deckhouse.io/dynamixinstanceclasses
    - deckhouse.io/gcpinstanceclasses
    - deckhouse.io/huaweicloudinstanceclasses
    - deckhouse.io/ingressistiocontrollers
    - deckhouse.io/istiofederations
    - deckhouse.io/istiomulticlusters
    - deckhouse.io/localpathprovisioners
    - deckhouse.io/openstackinstanceclasses
    - deckhouse.io/operationpolicies
```

```
    - deckhouse.io/projectnamespaces
    - deckhouse.io/projectrolebindings
    - deckhouse.io/projects
    - deckhouse.io/projecttemplates
    - deckhouse.io/securitypolicies
    - deckhouse.io/securitypolicyexceptions
    - deckhouse.io/vcdinstanceclasses
    - deckhouse.io/vsphereinstanceclasses
    - deckhouse.io/yandexinstanceclasses
    - deckhouse.io/zvirtinstanceclasses
    - expansion.gatekeeper.sh/expansiontemplate
    - externaldata.gatekeeper.sh/providers
    - limitranges
    - mutations.gatekeeper.sh/assign
    - mutations.gatekeeper.sh/assignimage
    - mutations.gatekeeper.sh/assignmetadata
    - mutations.gatekeeper.sh/modifyset
    - namespaces
    - network.deckhouse.io/metalloadbalancerbgppeers
    - network.deckhouse.io/metalloadbalancerclasses
    - network.deckhouse.io/metalloadbalancerconfigurations
    - network.deckhouse.io/metalloadbalancerpools
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - resourcequotas
    - sailoperator.io/istiocnis
    - sailoperator.io/istiorevisions
    - sailoperator.io/istiorevisiontags
    - sailoperator.io/istios
    - sailoperator.io/ztunnels
    - status.gatekeeper.sh/configpodstatuses
    - status.gatekeeper.sh/connectionpodstatuses
    - status.gatekeeper.sh/constraintpodstatuses
    - status.gatekeeper.sh/constrainttemplatepodstatuses
    - status.gatekeeper.sh/expansiontemplatepodstatuses
    - status.gatekeeper.sh/mutatorpodstatuses
    - status.gatekeeper.sh/providerpodstatuses
    - syncset.gatekeeper.sh/syncsets
    - templates.gatekeeper.sh/constrainttemplates
```

To get an additional list of access rules for a cluster module role (rules added by the administrator and non-default rules of other Deckhouse modules), use the following command:

```
D8_ROLE_NAME=Editor
d8 k get clusterrole -o jsonpath="{range .items[?(@.metadata.annotations.user-
authz\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

AuthorizationRule example

Use AuthorizationRule to set access rules for users within a specific namespace.

```
apiVersion: deckhouse.io/v1alpha1
kind: AuthorizationRule
metadata:
  name: test-rule
spec:
  accessLevel: Admin
  subjects:
  - kind: User
    name: admin@example.com
```

ClusterAuthorizationRule example

Use ClusterAuthorizationRule to set access rules for users either cluster-wide or for specific namespaces.

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
  - kind: User
    name: some@example.com
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci
  - kind: Group
    name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # This option is only available if the enableMultiTenancy parameter is set.
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: stage
        operator: In
        values:
        - test
        - review
      matchLabels:
        team: frontend
```

How rules are applied

ClusterAuthorizationRule and AuthorizationRule objects grant access through ordinary ClusterRoleBinding and RoleBinding objects. The `user-authz-controller` component creates these bindings for every rule, keeps them in sync with the rule, and writes the result to the rule's `status`. To view the state of the rules, run:

```
d8 k get clusterauthorizationrules
d8 k get authorizationrules -A
```

The `READY` column shows whether the bindings match the rule, and the `BINDINGS` column shows how many bindings the rule has. A rule with an unsupported `accessLevel` gets `Ready=False` with the `InvalidSpec` reason. The other reasons and the controller metrics are described in the `user-authz` module FAQ.

If several ClusterAuthorizationRule objects match the same subject, the allowed namespaces are combined, and the effective `accessLevel` is the most powerful among the matching rules. For an example, see the `user-authz` module FAQ.

Creating, updating, and deleting a ClusterAuthorizationRule counts as assigning roles: the request is admitted only if the requester can assign the roles of the rule. An update is checked against both the previous and the new grants, so lowering `SuperAdmin` to `User` still requires being able to assign `SuperAdmin`. Who can assign which levels is described in Who can assign roles.

A `User` or `Group` subject is matched by name against the identity in the token, and a local identity is indistinguishable from one asserted by an external authentication provider. Therefore, you cannot create a User whose `spec.email` or a Group whose `spec.name` matches a subject of an existing ClusterAuthorizationRule or AuthorizationRule. The restriction is checked when a User or Group is created and when its `spec.email` or `spec.name` is changed. If the match is intentional, for example, the rule was written in advance, set the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation on the User or Group. The annotation only acknowledges the name collision: it does not let the requester assign roles they cannot assign otherwise.

Extending permissions for high-level roles

To add permissions to a specific high-level role, create a ClusterRole with the annotation `user-authz.deckhouse.io/access-level: <AccessLevel>`.

The annotation accepts `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, and

`ClusterAdmin`. Other values, including `SuperAdmin`, are ignored. In the name of the aggregated role, the level is written in kebab case, for example, `user-authz:cluster-editor:custom`. The rules of a level are also included in the aggregated roles of every higher level.

Example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

Additional resources

- Module `user-authn` custom resources
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Granular authorization model

The granular role model is built on the principle of aggregation: it combines low-level roles into larger ones that cover common tasks. This makes the model easy to extend by adding your own roles.

To use the granular role model, the `user-authz` module must be enabled in the cluster. The module creates a set of special aggregated cluster roles (ClusterRole) suitable for most user and group access

management tasks.

⚠ The module implements two role-based models: the granular one (this page, recommended) and the basic one, built around the ClusterAuthorizationRule and AuthorizationRule resources (its support will be discontinued in future releases).

The models are not resource-compatible — automatic conversion is impossible — but they can be used at the same time: the permissions of both models are summed up.

Unlike the basic role model, the granular one does not use ClusterAuthorizationRule or AuthorizationRule resources. Access rights are configured the standard Kubernetes RBAC way: by creating RoleBinding or ClusterRoleBinding resources and specifying one of the roles prepared by the `user-authz` module in them. To grant access to all namespaces of a project at once, use the ProjectRoleBinding and ClusterProjectRoleBinding resources of the `multitenancy-manager` module.

i Access does not have to be granted by hand-writing YAML manifests: the Deckhouse web UI provides an access grant wizard. It walks you through the steps (who gets access → where → at which level), picks the right binding kind itself (RoleBinding, ClusterRoleBinding, ProjectRoleBinding, or ClusterProjectRoleBinding), and lets you assemble a custom role from ready-made building blocks without writing YAML.

The module creates special aggregated cluster roles (ClusterRole). By using these roles in RoleBinding or ClusterRoleBinding, you can do the following:

- Manage access to modules of a specific subsystem.

  For example, you can use the `d8:subsystem:network:manager` role in a ClusterRoleBinding to allow a network administrator to configure *network* modules (such as `cni-cilium`, `ingress-nginx`, `istio`, etc.).

- Manage access to *user* resources of modules within namespaces.

  For example, the `d8:namespace:manager` role in a RoleBinding enables deleting/creating/editing the PodLoggingConfig resource in the namespace. At the same time, it does not grant access to the cluster-wide ClusterLoggingConfig and ClusterLogDestination resources of the `log-shipper` module, nor does it allow configuration of the `log-shipper` module itself.

The roles created by the module are divided into the following classes:

- Namespace roles — for assigning rights to users (such as application developers) **in a specific namespace**.
- Project roles — for assigning rights **in all namespaces of a project at once**.
- System and subsystem roles — for assigning rights to platform administrators and administrators of a part of the platform.

Role scopes

Every role operates in one of four scopes. The scope defines *where* the granted permissions apply and *which resource* is used to assign the role:

Scope	Role name format	Intended for	Which resource is used to assign the role
Namespace	`d8:namespace:<level>`	Application users (developers)	RoleBinding in a specific namespace
Project	`d8:project:<level>`	Teams working with projects	ProjectRoleBinding or ClusterProjectRoleBinding for the whole project, RoleBinding for one namespace of the project
Subsystem	`d8:subsystem:<subsystem>:<level>`	Administrators of a part of the platform	ClusterRoleBinding
Whole platform	`d8:system:<level>`	Platform administrators	ClusterRoleBinding

Access levels

Each scope provides several access levels for roles:

- The “namespace” and “project” scopes have five levels: `viewer` → `user` → `manager` → `admin` → `superadmin`.
- The “subsystem” and “whole platform” scopes have three levels: `viewer` → `manager` → `superadmin`. There are no `user` and `admin` levels here: the system level has no “user” resources that could be used without administering them.

Access levels form a hierarchy with a cumulative principle: each next level inherits all permissions of the previous one and extends them. For example, the `manager` level in the namespace scope includes everything the `user` level allows, which in turn includes everything the `viewer` level allows.

Namespace roles

⚠ A namespace role cannot be assigned through a ClusterRoleBinding. Use a RoleBinding in a namespace, or a ProjectRoleBinding or ClusterProjectRoleBinding for all namespaces of a project.

Namespace roles are intended to assign rights to a user **in a specific namespace**. Users refer to, for example, developers who use a cluster configured by an administrator to deploy their applications. Such users don’t need to manage DP modules or a cluster, but they need to be able to, for example, create their Ingress resources, configure application authentication, and collect logs from applications.

A namespace role defines permissions for accessing namespaced resources of modules and standard namespaced resources of Kubernetes (Pod, Deployment, Secret, ConfigMap, etc.).

DP creates the following namespace roles:

Role	Allowed actions
`d8:namespace:viewer`	View standard Kubernetes resources (except Secrets and RBAC resources), Gateway API resources, pod logs and metrics, and authenticate in the cluster
`d8:namespace:user`	Everything `viewer` allows, plus: view secrets, `kubectl exec/attach`, delete pods (but not create or modify them), `kubectl port-forward/proxy`, change controller replica counts
`d8:namespace:manager`	Everything `user` allows, plus: manage module resources (for example, Certificate, PodLoggingConfig) and standard namespaced Kubernetes resources (Pod, Deployment, ConfigMap, Secret, Service, Ingress, NetworkPolicy, CronJob, etc.), as well as Gateway API resources (Gateway, ListenerSet, HTTPRoute, GRPCRoute, ReferenceGrant, etc.)
`d8:namespace:admin`	Everything `manager` allows, plus: manage ResourceQuota, LimitRange, ServiceAccount, Role, RoleBinding
`d8:namespace:superadmin`	Everything `admin` allows, plus security-sensitive operations: minting ServiceAccount tokens, making requests on behalf of ServiceAccounts

The detailed split of permissions between `admin` and `superadmin` is described in Admin level restrictions and superadmin rights.

Automatic access to cluster-wide catalog resources

Working in a namespace requires reading some cluster-wide “catalogs”: for example, to set `storageClassName` or `ingressClassName` in a manifest, one needs to see the list of StorageClasses and IngressClasses. Therefore, every subject that receives a RoleBinding to any `d8:namespace:*` or `d8:project:*` role automatically also gets read access to such catalog resources (StorageClass, IngressClass, PriorityClass, RuntimeClass, VolumeSnapshotClass, ClusterLogDestination, etc.).

Technically, this looks like an automatically created ClusterRoleBinding to the `d8:dict` role with the `rbac.deckhouse.io/dict: "true"` label. Such objects are managed by the platform: they appear when the subject is granted their first namespace binding and are removed once none are left — there is no need to edit them manually.

Project roles

> ⚠ A project role cannot be assigned via ClusterRoleBinding — such an attempt is rejected. To assign a role across a whole project, use ProjectRoleBinding or ClusterProjectRoleBinding. A plain RoleBinding in one of the project namespaces is also allowed — the role then applies in that namespace only.

Project roles (`d8:project:<level>`) are intended for working with projects — isolated environments that may span several namespaces. The levels are the same as for namespace roles: `viewer`, `user`, `manager`, `admin`, `superadmin`.

Each project role includes all permissions of the namespace role of the same level and additionally grants permissions to manage the project itself:

- `d8:project:viewer` — the permissions of `d8:namespace:viewer` plus viewing the project's ProjectNamespace and ProjectRoleBinding resources;
- `d8:project:user` — the permissions of `d8:namespace:user` plus viewing the project's ProjectNamespace and ProjectRoleBinding resources;
- `d8:project:manager` — the permissions of `d8:namespace:manager` plus managing the project's additional namespaces (ProjectNamespace resources);
- `d8:project:admin` — the permissions of `d8:namespace:admin` plus managing access to the project (ProjectRoleBinding resources) and the right to bind the built-in `d8:project:*` and `d8:namespace:*` roles (except the `superadmin` level) to other users within the project;
- `d8:project:superadmin` — analogous to the relation between `d8:namespace:superadmin` and `d8:namespace:admin`.

A role assigned via ProjectRoleBinding automatically applies in **all** namespaces of the project — the main one as well as the additional ones, including those created later.

System and subsystem roles

> ⚠ System and subsystem roles do not grant access to the namespaces of user applications.
>
> They grant access only to system namespaces (starting with `d8-` or `kube-`), and only to those system namespaces where the modules of the corresponding role subsystem are running.
>
> Whether a role lets you grant access to other people is described in Who can assign roles.

System and subsystem roles are intended for assigning rights to manage the entire platform or a part of it (the subsystem), but not the user applications themselves. The subsystem role, for example, can allow a security administrator to manage security modules (responsible for the security functions of the cluster). Thus, the security administrator will be able to configure security policies, certificate issuance, etc., but will not be able to manage other cluster functions (such as access management, network and monitoring settings) or change settings in the namespaces of user applications.

> ⚠ A system/subsystem role limits which modules and namespaces a subject can address, but it does not limit the privileges a subject can obtain through the modules it is allowed to manage. This matters most for the `iam` subsystem: the right to manage authentication and authorization is equivalent to full control over the cluster — a subject that can manage the `user-authn` module can register an identity provider or reset the credentials of any local user, and a subject that can manage the `user-authz` module can modify authorization rules. In both cases, they can obtain an identity with any privileges, including cluster administrator, so treat an `iam` subsystem role as a cluster administrator role when planning access.

The system/subsystem role defines access rights:

- to cluster-wide Kubernetes resources;
- to manage DP modules (ModuleConfig resource) within the subsystem of the role, or to all DP modules for the role `d8:system:*`;
- to manage cluster-wide resources of DP modules within the subsystem of the role, or to all resources of DP modules for the role `d8:system:*`;
- to read the namespaces of the modules of the role's subsystem, or of all DP modules for the role `d8:system:*`; writing there is reserved for the `superadmin` level (see Access of system and subsystem roles to the namespaces of the modules).

The role name format is `d8:system:<ACCESS_LEVEL>` for the system roles and `d8:subsystem:<SUBSYSTEM>:<ACCESS_LEVEL>` for the subsystem roles, where:

- `SUBSYSTEM` is the role's subsystem;
- `ACCESS_LEVEL` is the access level.

Role examples:

- `d8:system:viewer` — access to view the configuration of all DP modules (ModuleConfig resource), their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except Secrets and RBAC resources) in the namespaces of all DP modules;
- `d8:system:manager` — similar to `d8:system:viewer`, but with the right to view/create/modify/delete the configuration of all DP modules (ModuleConfig resource) and their cluster-wide resources; in the namespaces of DP modules it keeps the read access of `d8:system:viewer`;
- `d8:subsystem:observability:viewer` — access to view the configuration of DP modules (ModuleConfig resource) from the `observability` subsystem, their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except Secrets and RBAC resources) in the system namespaces `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, `kube-prometheus-pushgateway`.

DP has three access levels for system and subsystem roles:

- `viewer` — allows viewing standard Kubernetes resources, the configuration of modules (ModuleConfig resources), cluster-wide resources of modules, and namespaced resources of modules in the module namespace;
- `manager` — in addition to `viewer`, allows managing the configuration of modules (ModuleConfig resources) and the cluster-wide resources of modules; in the module namespaces it grants the same read access as `viewer`;
- `superadmin` — in addition to `manager`, allows everything in the module namespaces: managing standard Kubernetes resources and the namespaced resources of modules, reading Secrets, connecting to pods, issuing ServiceAccount tokens, and managing the system resources of the subsystem modules.

Access of system and subsystem roles to the namespaces of the modules

The holder of a system or subsystem role automatically gets a RoleBinding to a namespace role in every system namespace in which the modules of the role's subsystem run (a `d8:system:*` role covers the namespaces of all DP modules): the `viewer` and `manager` levels get `d8:namespace:viewer`, the `superadmin` level gets `d8:namespace:superadmin`. The platform creates and removes these RoleBindings itself (they carry the `rbac.deckhouse.io/automated: "true"` label), one per ClusterRoleBinding of the role.

The namespaces of the modules belong to the platform. They hold ServiceAccounts with cluster-wide rights (the Deckhouse controller among them) and the credentials of the modules, so any write there, and any read of a Secret there, is in effect the right to act as those ServiceAccounts anywhere in the cluster. This is why a `manager` configures its subsystem through ModuleConfig objects and the cluster-wide resources of the modules, and in the module namespaces only views objects, logs, metrics, and events. A `manager` cannot read Secrets, run `kubectl exec`, `attach`, or `port-forward`, delete or scale pods, or create anything there. A `superadmin` can do all of that. Grant this level deliberately and to a named subject: it is the emergency path into the platform, and its actions are visible in the audit log. Do not use impersonation of system users (`--as`) as a substitute.

When working with system and subsystem roles, keep in mind the following:

- A RoleBinding to `d8:namespace:user` or a higher level in a system namespace, created by hand, grants the subject the rights of every ServiceAccount of that namespace. The same holds for the `allowAccessToSystemNamespaces: true` option of a ClusterAuthorizationRule of the basic role model: with it, the `Admin` and `ClusterAdmin` access levels are equivalent to a cluster administrator.
- The system namespaces are those the `multitenancy-manager` module never turns into projects: `d8-*`, `kube-*`, `upmeter-*`, `default`, and the namespaces labeled `heritage: deckhouse` or `heritage: upmeter`.
- A custom system or subsystem role (`d8:custom:*`) is projected as `viewer` whatever its `rbac.deckhouse.io/use-role` label says: the webhook refuses a higher value, and a role created with one earlier is projected as `viewer` with a `ProjectionCapped` event on the role.

Who can assign roles

A system or subsystem role does not, by itself, let you grant access to other people.

Creating a User or Group that is not already a grant subject is ordinary object creation.

Creating a User for an email that already carries a grant, writing or deleting a ClusterAuthorizationRule, or resetting that identity through UserOperation, is granting roles. The request is admitted only if the

requester already has covering permissions or is explicitly allowed to assign those roles.

Connecting a DexProvider is measured the same way against the identities the provider can assert: `spec.allowedIdentities` (emails, email domains, groups) and the provider's own group filters bound that set, and only the roles already granted inside it count. A provider without an email limit can assert every email, including the ones that hold `SuperAdmin`, so writing it requires `SuperAdmin`. Basic Authentication of an OIDC or a Crowd provider bypasses Dex and its limits, so such a provider counts as one without an email limit, and an OIDC one also without a group limit; a Crowd provider keeps only `crowd.groups` on the group axis. Details and the list of changes admitted without a check are in the `user-authn` module documentation.

The ClusterAuthorizationRule `spec.accessLevel` field is a basic model level: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`. An `iam` subsystem manager and a holder of `d8:system:manager` assign the levels `ClusterEditor` includes: `User`, `PrivilegedUser`, `Editor`, and `ClusterEditor`, but not `Admin`, `ClusterAdmin`, or `SuperAdmin`, and neither the Kubernetes `cluster-admin` ClusterRole.

A ClusterAuthorizationRule that opens the system namespaces counts as a `ClusterAdmin` assignment whatever its `accessLevel`, so these managers cannot write it. Such a rule is a rule with `allowAccessToSystemNamespaces: true`, with `namespaceSelector.matchAny`, with a `namespaceSelector.labelSelector` that does not name its namespaces by the `kubernetes.io/metadata.name` label, a rule for a subject that another rule already lets into them (the rules of a subject add up), or any rule while multitenancy mode is disabled. Without multitenancy mode, as in DP Open/CE, these managers therefore cannot write ClusterAuthorizationRule objects at all.

The rules of a request add up: those of its user, of every group in its token, and of its ServiceAccount. A rule therefore also opens the system namespaces when another rule already lets in one of the requests it applies to:

- a local user it names or a member of its groups at any depth of nesting, with the groups that user is in and the groups above them;
- the requester, with the groups of its own token;
- any request, through `system:authenticated` (a ServiceAccount also through `system:serviceaccounts` and `system:serviceaccounts:<namespace>`).

A rule for `system:authenticated` applies to every request, so it opens the system namespaces as soon as any other rule opens them. A rule for `system:serviceaccounts` does so as soon as another rule lets in a ServiceAccount, and a rule for `system:serviceaccounts:<namespace>` as soon as another rule lets in a ServiceAccount of that namespace. A User subject named `system:serviceaccount:<namespace>:<name>` is that ServiceAccount.

A member of a Group gets the rules of every group above it, so writing a Group needs the range for their roles too. For the same reason, these managers cannot add a member that another rule already lets into the system namespaces to a Group that has a ClusterAuthorizationRule, or is in a group that has one. The groups an identity provider asserts for other users are unknown to the webhook and are not taken into account.

For the same reason, a user or a group that such a rule names counts as a `ClusterAdmin` target: these managers cannot create a User for its email, add members to such a Group, run a UserOperation on such a user, or connect a DexProvider that can assert such an identity. Turning multitenancy mode off in the `user-authz` ModuleConfig opens the system namespaces to every rule and requires the `ClusterAdmin` range as well. Within the granular model, the IAM manager assigns the `iam` subsystem roles up to `d8:subsystem:iam:manager`, and the system manager assigns the roles of any scope up to the `manager` level.

These managers do not write admission webhooks and policies (ValidatingWebhookConfiguration, MutatingWebhookConfiguration, ValidatingAdmissionPolicy, and ValidatingAdmissionPolicyBinding): the superadmins of the `security` and `cluster` subsystems and `d8:system:superadmin` do. A mutating webhook can change any object the platform creates, its own pods included, and removing a webhook or a policy switches off what it enforces, such as the limits above. The `d8-deckhouse-validating-webhook-handler-*` ValidatingWebhookConfigurations, which hold the webhooks of the Deckhouse modules, can be created, changed, or deleted only by `webhook-handler`, the platform components, and the superadmins, including the holders of a role that grants every verb on everything.

A requester with every permission across the cluster counts as `SuperAdmin` in these checks and can assign any role, including `SuperAdmin` and `cluster-admin`. For example, a subject of the Kubernetes `cluster-admin` ClusterRole has such permissions if the role is granted by a ClusterRoleBinding or by a ClusterAuthorizationRule without `namespaceSelector` and `limitNamespaces`. If multitenancy mode is enabled, such a rule reaches the system namespaces only with `namespaceSelector.matchAny: true` (or the deprecated `allowAccessToSystemNamespaces: true`).

Permissions from a ClusterAuthorizationRule with `namespaceSelector` or `limitNamespaces` do not count toward this, even if the rule grants `cluster-admin`. Any non-empty `limitNamespaces` counts as a limit on purpose, even if its entries match every namespace, such as `.*` together with `allowAccessToSystemNamespaces: true`. Such a rule still counts through its `accessLevel`. For

example, `accessLevel: SuperAdmin` gives the `SuperAdmin` range whatever the namespace limits of the rule.

`limitNamespaces` and `namespaceSelector` restrict access to namespaced resources only. A subject with the `ClusterAdmin` or `SuperAdmin` access level can still create ClusterRoleBinding and ClusterAuthorizationRule objects, so these fields do not confine what it can grant across the cluster. Give these levels only to subjects that may hold access to the whole cluster. A ClusterAuthorizationRule counts only through the bindings that `user-authz-controller` has created for it.

Webhook deny messages call the `accessLevel` values basic, so they are not confused with the granular role levels (`viewer`, `manager`, `superadmin`).

Permission to create User and Group objects in the `user-authn` module is not enough: it does not grant the roles already attached to that email.

Global resource dictionaries

In addition to the roles and capabilities of the granular model, the module creates a special ClusterRole `d8:dict`. It grants read access to cluster-wide “reference” resources that users often need to view when creating objects. For example, a user creating a PersistentVolumeClaim needs to see the available StorageClasses, and a user creating an Ingress needs to see the IngressClasses.

The role grants `get`, `list`, `watch` on the following resources:

- `storageclasses` (storage.k8s.io), plus `csidrivers`, `csinodes`, `volumeattachments`;
- `volumesnapshotclasses` (snapshot.storage.k8s.io);
- `ingressclasses` (networking.k8s.io);
- `priorityclasses` (scheduling.k8s.io);
- `runtimeclasses` (node.k8s.io);
- `virtualmachineclasses`, `clustervirtualimages` (virtualization.deckhouse.io);
- `clusterlogdestinations` (deckhouse.io);
- `customresourcedefinitions` (apiextensions.k8s.io) — `get`, `list` only.

Automatic binding

`user-authz-controller` keeps one ClusterRoleBinding to `d8:dict` for every subject that holds one of the following RoleBindings:

- a RoleBinding to a `d8:namespace:*` or `d8:project:*` role, including the RoleBindings that `multitenancy-manager` creates from a ProjectRoleBinding or a ClusterProjectRoleBinding;
- a RoleBinding that the module creates for an AuthorizationRule (the `User`, `PrivilegedUser`, `Editor`, and `Admin` levels).

The `system:multitenancy-manager:*` users, which `multitenancy-manager` acts as, get none. This means that project users can discover reference resources without any manual configuration. The binding is removed when the subject has no such RoleBinding left. You do not create it yourself.

> **i** The `d8:dict` role is independent of the `multitenancy-manager` cluster-wide resource access management mechanism: it only gives **read** access to reference resources so users can discover them. The mechanism controls **which resource values** a project may actually reference when creating objects.

Role model subsystems

Each DP module belongs to a specific subsystem. For each subsystem, there is a set of roles with different levels of access. Roles are updated automatically when the module is enabled or disabled.

The subsystems of the platform are `iam`, `security`, `cluster`, `delivery`, `network`, `storage`, `observability`, and `managed-services`.

For example, for the `network` subsystem, there are the following subsystem roles that can be used in ClusterRoleBinding:

- `d8:subsystem:network:viewer`;
- `d8:subsystem:network:manager`;
- `d8:subsystem:network:superadmin`.

The scope of a role depends on which subsystem it belongs to:

- The scope of the `d8:system:*` roles is the namespaces of all enabled DP modules, that is, the namespaces that their capabilities name in the `rbac.deckhouse.io/namespace` label.
- The scope of subsystem roles includes the namespaces in which the subsystem’s modules operate

(see the subsystem composition table below), as well as all cluster-wide objects of the subsystem’s modules.

Role model subsystems composition table.

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
all	All modules	All namespaces
iam	• multitenancy-manager • namespace-configurator • user-authn • user-authz	• d8-multitenancy-manager • d8-system • d8-user-authn
security	• admission-policy-engine • cert-manager • neuvector • operator-trivy • runtime-audit-engine • secret-copier • secrets-store-integration	• d8-admission-policy-engine • d8-cert-manager • d8-operator-trivy • d8-runtime-audit-engine
cluster	• chrony • cloud-provider-aws • cloud-provider-azure • cloud-provider-dynamix • cloud-provider-gcp • cloud-provider-huaweicloud • cloud-provider-openstack • cloud-provider-vcd • cloud-provider-vsphere • cloud-provider-yandex • cloud-provider-zvirt • commander-agent • console • control-plane-manager • csi-vsphere • dashboard • deckhouse • deckhouse-tools • descheduler • documentation • node-manager • pod-reloader • registry • registry-packages-proxy • terraform-manager • vertical-pod-autoscaler • virtualization	• d8-chrony • d8-cloud-instance-manager • d8-cloud-provider-aws • d8-cloud-provider-azure • d8-cloud-provider-dynamix • d8-cloud-provider-gcp • d8-cloud-provider-huaweicloud • d8-cloud-provider-openstack • d8-cloud-provider-vcd • d8-cloud-provider-vsphere • d8-cloud-provider-yandex • d8-cloud-provider-zvirt • d8-csi-vsphere • d8-descheduler • d8-system • kube-system
delivery	• delivery-kit • operator-helm	• d8-operator-helm

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
network	• cilium-hubble • cni-cilium • cni-flannel • cni-simple-bridge • ingress-nginx • istio • keepalived • kube-dns • kube-proxy • metallb • monitoring-ping • network-gateway • network-policy-engine • node-local-dns • openvpn • sdn • service-with-healthchecks • static-routing-manager	• d8-cni-cilium • d8-cni-flannel • d8-cni-simple-bridge • d8-ingress-nginx • d8-istio • d8-keepalived • d8-metallb • d8-monitoring • d8-network-gateway • d8-openvpn • d8-service-with-healthchecks • d8-static-routing-manager • d8-system • kube-system
storage	• csi-ceph • csi-hpe • csi-huawei • csi-netapp • csi-nfs • csi-s3 • csi-scsi-generic • csi-yadro-tatlin-unified • local-path-provisioner • sds-local-volume • sds-node-configurator • sds-replicated-volume • snapshot-controller • storage-volume-data-manager	
observability	• extended-monitoring • log-shipper • loki • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-kubernetes-control-plane • observability • okmeter • operator-prometheus • prometheus • prometheus-metrics-adapter • prometheus-pushgateway • upmeter	• d8-log-shipper • d8-monitoring • d8-okmeter • d8-operator-prometheus • d8-upmeter • kube-prometheus-pushgateway
managed-services	• managed-memcached • managed-postgres	

How the roles are built: aggregation and capabilities

No built-in role contains a list of permissions directly. Permissions are described in separate small cluster roles — **capabilities**. Each capability is responsible for one kind of action (for example, “view logs”, “manage quotas”, “connect to pods”) and contains concrete RBAC rules. A role (`d8:namespace:admin`, `d8:system:viewer`, etc.) is an empty ClusterRole with an aggregation rule (`aggregationRule`): Kubernetes automatically collects into it the rules from all capabilities with matching labels.

This design has two practical consequences:

- DP modules extend the roles automatically: when a module is enabled, its capabilities are added to the corresponding built-in roles; when it is disabled, they are removed. The permission list of a role always matches the set of enabled modules.
- You can assemble your own roles from ready-made capabilities without writing RBAC rules by hand, as shown below.

The names of built-in roles and capabilities start with the `d8:` prefix. This prefix is reserved: you cannot create your own ClusterRole with a `d8:*` name — the only exception is the `d8:custom:*` prefix, which is dedicated to user-defined roles and capabilities.

Creating a new subsystem role

If, for example, the current subsystems do not fit the role distribution in the company, you need to create a new subsystem that includes roles from the `cluster` subsystem, the `network` subsystem, and the `user-authn` module.

To meet this need, create the following role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: viewer
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-cluster-as: manager
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-network-as: manager
    - matchLabels:
        rbac.deckhouse.io/scope: system
        module: user-authn
rules: []
```

The labels for the new role listed at the top suggest that:

- The namespace role that the subjects of the role get in the module namespaces (`user-authz-controller` creates the RoleBinding objects); a custom role is projected as `viewer` at most:

  ```
  rbac.deckhouse.io/use-role: viewer
  ```

- The role is a custom role (custom roles never define their own rules, they only aggregate capabilities):

  ```
  rbac.deckhouse.io/kind: custom-role
  ```

  > **i** This label is mandatory.

- The role is a subsystem one, and it shall be handled accordingly:

  ```
  rbac.deckhouse.io/scope: subsystem
  ```

- There is a subsystem for which the role is responsible:

  ```
  rbac.deckhouse.io/subsystem: mycustom
  ```

Then there are selectors that implement aggregation:

- This one aggregates the manager role from the `cluster` subsystem:

  ```
  rbac.deckhouse.io/aggregate-to-cluster-as: manager
  ```

- This one aggregates all the system-scope capabilities defined for the `user-authn` module:

  ```
  rbac.deckhouse.io/scope: system
  module: user-authn
  ```

> **i** For more information on role labels and annotations, see Reference of role labels and annotations.

This way, your role will combine the permissions of the `cluster` subsystem, `network` subsystem, and the `user-authn` module.

Specifics of using custom roles:

- Custom roles and capabilities must be named with the `d8:custom:` prefix (all other names with the `d8:` prefix are reserved for DP built-in objects). The name must agree with the declared scope: a subsystem role is `d8:custom:<subsystem>:<name>` (the segment is the subsystem itself, as in the example above), a namespace or project role is `d8:custom:namespace:<name>` or `d8:custom:project:<name>`, and a capability is `d8:custom:<scope>-capability:<name>`. A name that disagrees with the `rbac.deckhouse.io/scope` label is rejected.
- RoleBindings with a namespace role (`d8:namespace:<level>`) will be created in the namespaces of the aggregated subsystems' modules; the level is specified by the `rbac.deckhouse.io/use-role` label, `viewer` at most for a custom role. Without the label, no such RoleBinding is created.
- A custom role cannot be aggregated into `d8:system:<level>`, because only a custom capability may carry the `rbac.deckhouse.io/aggregate-to-system-as` label. To give the role to the holders of `d8:system:manager`, bind it to them with a ClusterRoleBinding.

Extending a custom role

Suppose a new cluster CRD object, MySuperResource, has been created in the cluster (a subsystem role example), and you need to extend the custom role from the example above to include the permissions to interact with this resource. Follow these steps:

1. Add a new selector to the role:

   ```
   rbac.deckhouse.io/aggregate-to-mycustom-as: manager
   ```

   For more information on role labels and annotations, see Reference of role labels and annotations.

   This selector enables capabilities to be aggregated to the new subsystem by specifying this label. After adding the new selector, the role looks as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: viewer
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-cluster-as: manager
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-network-as: manager
    - matchLabels:
        rbac.deckhouse.io/scope: system
        module: user-authn
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-mycustom-as: manager
rules: []
```

2. Create a new capability that defines the permissions for the new resource. For example, read-only access:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-mycustom-as: manager
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
    rbac.deckhouse.io/capability: "custom.subsystem-
capability.mycustom.superresource_view"
  name: d8:custom:subsystem-capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

   The capability adds its permissions to the subsystem role, granting the right to view the new object.

Specifics:

- Name a custom capability `d8:custom:<SCOPE>-capability:<NAME>`, where `<SCOPE>` is the value of its `rbac.deckhouse.io/scope` label. A name that disagrees with the label is rejected.

Extending existing subsystem roles

To extend an existing role, follow the same steps as in the section above, but change the labels and the role name. For more information on role labels and annotations, see Reference of role labels and annotations.

For example, here’s how you can extend the manager role from the `cluster` (`d8:subsystem:cluster:manager`) subsystem:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-cluster-as: manager
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: cluster
    rbac.deckhouse.io/capability: "custom.subsystem-
capability.cluster.superresource_view"
  name: d8:custom:subsystem-capability:cluster:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This way, the new capability will extend the `d8:subsystem:cluster:manager` role.

Extending subsystem roles by adding a new namespace

The namespaces where the holders of a system or subsystem role get a RoleBinding with the namespace role are the namespaces of its modules. `user-authz-controller` takes them from the `rbac.deckhouse.io/namespace` label of the system capabilities Deckhouse ships, which have a `d8:` name and the `heritage: deckhouse` label. On a custom capability, the label is ignored: the capability adds its rules to the roles that aggregate it, but no namespace.

To give the subjects of such a role access to another namespace, bind them to a namespace role there with a RoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cluster-managers-viewer
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: d8:namespace:viewer
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: cluster-managers
```

Extending existing namespace roles

If the resource belongs to a namespace, you need to extend the namespace role instead of the system/subsystem role. The only difference is the labels and the name:

```
 apiVersion: rbac.authorization.k8s.io/v1
 kind: ClusterRole
 metadata:
   labels:
     rbac.deckhouse.io/aggregate-to-namespace-as: user
     rbac.deckhouse.io/kind: custom-capability
     rbac.deckhouse.io/scope: namespace
     rbac.deckhouse.io/capability: "custom.namespace-
capability.mycustom.superresource_view"
   name: d8:custom:namespace-capability:mycustom:superresource:view
 rules:
 - apiGroups:
   - mygroup.io
   resources:
   - mysuperresources
   verbs:
   - get
   - list
   - watch
```

This capability will be added to the `d8:namespace:user` role.

For more information on role labels and annotations, see Reference of role labels and annotations.

Creating a custom namespace or project role

Sometimes the built-in hierarchy of levels does not fit: for example, you need a “developer” role — full view of the namespace plus terminal access to pods (`exec`, `attach`), but without the right to change quotas or RBAC. Such a role is assembled from ready-made capabilities, without writing RBAC rules by hand.

The rules for custom roles:

- the name must start with `d8:custom:` (for example, `d8:custom:namespace:developer`);
- the role must carry the `rbac.deckhouse.io/kind: custom-role` label;
- a namespace or project role that will be used in a RoleBinding must also carry `rbac.deckhouse.io/delegatable: "true"`. Every user namespace is a project, and a RoleBinding there is accepted only for roles with this label. Do not put it on system or subsystem roles — the admission webhook rejects that;
- the role **cannot contain its own rules** (`rules`) — it may only aggregate capabilities via `aggregationRule`. Permissions are described in separate capabilities, so the contents of the role stay transparent;
- a single role cannot aggregate capabilities of the user-facing scopes (`namespace`, `project`) together with the administrative ones (`system`, subsystems) — such a role is rejected.

For more information on role labels and annotations, see Reference of role labels and annotations.

An example: a role that includes everything allowed by `d8:namespace:viewer`, plus one specific capability (connecting to pods), selected precisely by its unique `rbac.deckhouse.io/capability` label:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace:developer
  labels:
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/delegatable: "true"   # Required for a RoleBinding inside a
project namespace.
  annotations:
    custom.meta.deckhouse.io/title: "Developer"
    custom.meta.deckhouse.io/description: "View resources and connect to pods,
without managing quotas and RBAC"
aggregationRule:
  clusterRoleSelectors:
    # Everything included in the viewer level of the namespace lineage.
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-namespace-as: viewer
    # Plus one specific capability selected by its unique name.
    - matchLabels:
        rbac.deckhouse.io/capability: "namespace-
capability.kubernetes.access_terminal"
rules: []
```

If no existing capability grants the permissions you need, create your own (a `custom-capability` may contain rules) and add a selector by its `rbac.deckhouse.io/capability` label to the role's `aggregationRule` (in the example below — `matchLabels: {rbac.deckhouse.io/capability: "custom.namespace-capability.port-forward"}`). The label value of a custom capability is `custom.<SCOPE>-capability.<NAME>`, and a role that is bound in a project may select only the `namespace-capability` and `project-capability` scopes:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace-capability:port-forward
  labels:
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/capability: "custom.namespace-capability.port-forward"
rules:
  - apiGroups: [""]
    resources: ["pods/portforward"]
    verbs: ["create"]
```

To get a list of all available capabilities and their unique names, use the command:

```
d8 k get clusterroles -l rbac.deckhouse.io/kind=capability -o custom-
columns='NAME:.metadata.name,CAPABILITY:.metadata.labels.rbac\.deckhouse\.io/capabili
ty'
```

The created role is assigned exactly like a built-in one: via a RoleBinding in a namespace or via a ProjectRoleBinding across a whole project (for project roles, use `rbac.deckhouse.io/scope: project` and aggregate `aggregate-to-project-as`). A RoleBinding in a project namespace requires the `delegatable` label on the role; a ProjectRoleBinding does not (it has its own prefix allow-list). The role cannot be assigned via a ClusterRoleBinding — just like the built-in roles of these scopes.

> **i** You can also assemble such a role without YAML — with the access grant wizard in the Deckhouse web UI: it shows the available capabilities, builds a role out of them, and immediately creates the required binding.

Admin level restrictions and superadmin rights

The role model provides two administration levels:

- `admin` — the everyday administrator. Manages resources, quotas, and access within their scope, but cannot perform operations that would let them break out of that scope or disrupt platform components.
- `superadmin` — the “break-glass” administrator. Has all the rights of `admin` and can additionally perform dangerous operations. Grant this level deliberately and only to those who really need it.

Only the `superadmin` level allows **minting ServiceAccount tokens** (`kubectl create token`) **and making requests on behalf of a ServiceAccount** (`kubectl --as system:serviceaccount:...`). A ServiceAccount token is a ready-to-use identity: by obtaining the token of a platform component’s service account, one could gain its permissions far beyond the namespace. Therefore `admin` manages the `ServiceAccount` objects themselves (create, delete) but cannot mint tokens for them or act on their behalf.

Objects labeled `heritage: deckhouse` belong to the platform: nobody may create, modify or delete them, `superadmin` and cluster administrators included. Some platform components place such objects in user namespaces, for example the pods of a DexAuthenticator. `kubectl exec`, `kubectl attach`, `kubectl port-forward` and `kubectl debug` into such pods and `kubectl scale` of such Deployment, StatefulSet, ReplicaSet and ReplicationController objects are denied too. `kubectl rollout restart` of such objects is allowed. The users these checks do not apply to are listed in the FAQ of the `deckhouse` module.

`superadmin` also has some restrictions: resources created from a project template (the `heritage: multitenancy-manager` label) cannot be modified by **anyone**, including `superadmin` and a cluster administrator — they are managed exclusively by the project controller. A role is assigned via RoleBinding and applies only in the namespace where it was granted: a `superadmin` of one namespace gets no special rights in another.

Built-in protection mechanisms of the role model

The role model is protected by a set of checks at the API server level. They require no configuration and prevent typical mistakes and privilege escalation attempts:

- **A scoped role cannot be granted cluster-wide.** A ClusterRoleBinding to the `d8:namespace:*` or `d8:project:*` roles (and their `d8:custom:*` variants) is rejected — otherwise a role designed for one namespace or project would apply in every namespace at once. Use a RoleBinding in the desired namespace, or ProjectRoleBinding/ClusterProjectRoleBinding for a project.
- **A capability cannot be granted cluster-wide.** A ClusterRoleBinding to any capability is rejected: a capability is a building block for roles, not a standalone role. Binding a capability via a RoleBinding in a single namespace is allowed.
- **Project management cannot be obtained through a custom role.** Creating a Role or ClusterRole that grants permissions to modify project management resources (`projects`, `projecttemplates`, `projectrolebindings`, `clusterprojectrolebindings`, `projectnamespaces`) is rejected — these permissions are granted only by the built-in `d8:project:*` roles.
- **A namespace or project role cannot include platform capabilities.** A custom role with `scope: namespace`, `scope: project`, or no scope may not select system or subsystem capabilities by the `rbac.deckhouse.io/capability` label, nor the `system` or a subsystem scope by the `rbac.deckhouse.io/aggregate-to-<scope>-as` label. No custom role may combine the capabilities of the `namespace`/`project` scopes with those of the `system`/`subsystem` scopes.
- **A custom capability carries rules and cannot have an `aggregationRule`.**
- **The name must match the scope**: `d8:custom:namespace:<name>`, `d8:custom:project:<name>`, `d8:custom:system:<name>`, `d8:custom:<subsystem>:<name>`, and `d8:custom:<scope>-capability:<name>` for a capability.
- **Only a custom capability may carry an `rbac.deckhouse.io/aggregate-to-<scope>-as` label of a platform scope.**
- **The `rbac.deckhouse.io/delegatable: "true"` label is accepted only on a role that aggregates only namespace and project capabilities.**
- **Custom roles cannot contain direct RBAC rules** — they may only aggregate capabilities.
- **The `rbac.deckhouse.io/kind: role` and `rbac.deckhouse.io/kind: capability` labels are reserved for DP built-in objects.** A ClusterRole with one of these labels is rejected. For your own objects, use `custom-role` and `custom-capability`.

Display names of roles

Every built-in role and capability carries a localized title and description in annotations:

- `ru.meta.deckhouse.io/title`, `ru.meta.deckhouse.io/description` — in Russian;
- `en.meta.deckhouse.io/title`, `en.meta.deckhouse.io/description` — in English.

These annotations are used, for example, by the Deckhouse web UI when displaying the list of roles.

If the standard title does not fit (for example, you want to name roles in your company's terms), add the `custom.meta.deckhouse.io/title` and `custom.meta.deckhouse.io/description` annotations to the role — the interface will show them instead of the standard ones. Example:

```
d8 k annotate clusterrole d8:namespace:admin custom.meta.deckhouse.io/title='Team administrator'
```

This is the only allowed modification of built-in roles: changing their rules, aggregation, or labels is rejected.

Deprecated role names

The previous role names of the granular model (`d8:manage:<subsystem>:<level>`, `d8:manage:all:<level>`, and `d8:use:role:<level>`) are deprecated and will be removed in the next release. For backward compatibility they are temporarily kept as alias roles. A binding to an alias aggregates the **new** role's capabilities, except where the list of subsystems changed (see below).

The rights of an alias are not identical to the pre-migration ones. The deprecated `d8:use:role:admin` no longer grants ServiceAccount token minting or impersonation, which moved to `superadmin`. The `d8:manage:*` aliases are not `delegatable`. The `d8:use:role:*` aliases are, so a RoleBinding to them inside a project namespace keeps working, but a new ProjectRoleBinding or ClusterProjectRoleBinding to an old name is refused.

Name mapping:

Deprecated name	New name
`d8:manage:all:<level>`	`d8:system:<level>`
`d8:manage:deckhouse:<level>`, `d8:manage:infrastructure:<level>`, `d8:manage:kubernetes:<level>`	`d8:subsystem:cluster:<level>`, which grants more. The modules that moved to other subsystems come with the roles of those subsystems
`d8:manage:networking:<level>`	`d8:subsystem:network:<level>`, which also grants the rights for `keepalived`
`d8:manage:security:<level>`	`d8:subsystem:security:<level>` together with `d8:subsystem:iam:<level>`, which grants the rights for `user-authz` and `user-authn` and more
`d8:manage:observability:<level>`	`d8:subsystem:observability:<level>`
`d8:manage:storage:<level>`	`d8:subsystem:storage:<level>`
`d8:use:role:<level>`	`d8:namespace:<level>`
`d8:use:role:<level>:kubernetes`	`d8:namespace:<level>` (full namespace role, not kubernetes-only)

The list of subsystems changed together with the role names. `deckhouse`, `infrastructure`, and `kubernetes` were merged into `cluster`, which also took over modules from other subsystems, `networking` was renamed to `network`, and `user-authz`, `user-authn`, and `multitenancy-manager` moved to `iam`.

The aliases of the `deckhouse`, `infrastructure`, and `kubernetes` roles therefore do not aggregate the whole `cluster` subsystem. They select the capabilities of the platform modules that were in the old subsystem, wherever those modules are now. `d8:subsystem:cluster:<level>` grants more, so when you recreate such a binding, check which subsystem roles its subjects need.

The alias of `d8:manage:security:<level>` also keeps the rights for `user-authz` and `user-authn`, which now come with `d8:subsystem:iam:<level>`. That role also grants the rights for `multitenancy-manager` and, at the `manager` level, the right to assign roles, which `d8:subsystem:security:<level>` does not grant. So a binding to `d8:manage:security:<level>` is replaced by two bindings with the same subjects, to `d8:subsystem:iam:<level>` and to `d8:subsystem:security:<level>`. The `rbac.deckhouse.io/deprecated-replaced-by` annotation of the alias names only the second one.

The alias of `networking` leaves out `keepalived`, which moved to `network` from `infrastructure`, so the new role grants more than this alias.

The `rbac.deckhouse.io/deprecated-replaced-by` annotation of an alias names its main replacement, and the description of the alias names the roles of the other subsystems its modules moved to. These roles cover whole subsystems. To grant the rights for some modules only, create a custom role that selects their system capabilities by the `rbac.deckhouse.io/capability` label, for

example `system-capability.multitenancy-manager.view` (see Creating a new subsystem role).

The aliases of the `deckhouse`, `infrastructure`, `kubernetes`, and `networking` roles and `d8:manage:all:<level>` also select the labels of these old subsystems, such as `rbac.deckhouse.io/aggregate-to-deckhouse-as`. Modules released on their own and custom capabilities may still carry them. No role of the new model collects these labels, so such capabilities reach the holders of the old names only, and the `D8UserAuthzRetiredAggregationLabel` alert names them.

As long as bindings to the deprecated names remain in the cluster, the `D8UserAuthzDeprecatedRBACv2RoleInUse` (a binding to an alias role) and `D8UserAuthzDeprecatedRBACv2CapabilityInUse` (a binding to a deprecated name that no ClusterRole has, such as a capability of the previous release, so the binding grants nothing) alerts fire. The release that removes the aliases carries the `deprecatedRBACv2BindingsCount` release requirement: the upgrade stays pending, with the offending bindings listed in the DeckhouseRelease status, until every binding to an alias role is recreated on a new role name. A binding that grants nothing does not hold the release.

Migrate your existing RoleBinding and ClusterRoleBinding objects to the new role names:

- A ClusterRoleBinding named `user-authz:<rule>:additional-role:<role>` is rendered by `user-authz-controller` from `spec.additionalRoles` of the ClusterAuthorizationRule `<rule>`, and the controller creates it again when it is deleted. Change the role name in `spec.additionalRoles` of that rule, not the binding.
- A binding that a project template renders is changed in the ProjectTemplate, because `multitenancy-manager` renders it again from the template.

You can find bindings that still use the deprecated names, without the RoleBindings that `user-authz-controller` keeps for them, with the following command:

```
d8 k get clusterrolebindings,rolebindings -A -o json \
  | jq -r '.items[] | select(.roleRef.name | test("^d8:(manage|use):")) |
select(.metadata.labels["rbac.deckhouse.io/automated"] != "true") | "\(.kind) \
(.metadata.namespace // "-") \(.metadata.name) -> \(.roleRef.name)"'
```

Custom roles of the previous scheme

> ⚠ The built-in capabilities no longer carry the `rbac.deckhouse.io/kind: manage` and `rbac.deckhouse.io/kind: use` labels. A custom role of the previous scheme that selects capabilities by these labels has lost their permissions, and so have its subjects. Custom roles get no compatibility aliases.

The `D8UserAuthzLegacyRBACv2CustomRoleFound` alert lists such roles. How to find and repair them, along with the mapping of the previous labels to the new ones, is described in the `user-authz` FAQ section on migrating custom roles to the new scheme.

Getting an equivalent of the basic model's ClusterAdmin and SuperAdmin roles

The granular model has no roles that combine permissions into a single entity, like the `ClusterAdmin` and `SuperAdmin` roles of the basic model. It separates platform administration (system roles) from application access (namespace and project roles). The equivalent is assembled from **two bindings**: a ClusterRoleBinding to a system role and a ClusterProjectRoleBinding to a project role (the latter applies in all projects, including those created later).

Approximate level mapping:

Basic model role	Granular model equivalent
`User`	`d8:namespace:viewer` (via RoleBinding or ProjectRoleBinding)
`PrivilegedUser`	`d8:namespace:user`
`Editor`	`d8:namespace:manager`
`Admin`	`d8:namespace:admin`

Basic model role	Granular model equivalent
`ClusterEditor`	No direct counterpart: assemble it from a ClusterProjectRoleBinding to `d8:project:manager` (all projects) plus a system role for the platform part
`ClusterAdmin`	`d8:system:manager` + a ClusterProjectRoleBinding to `d8:project:admin`
`SuperAdmin`	`d8:system:superadmin` + a ClusterProjectRoleBinding to `d8:project:superadmin`

An example for `ClusterAdmin` (the `k8s-admins` group):

```
# Platform: DP module configuration, cluster-wide resources, viewing the namespaces
of DP modules.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: k8s-admins-platform
subjects:
  - kind: Group
    name: k8s-admins
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:system:manager
  apiGroup: rbac.authorization.k8s.io
---
# Applications: administrator in all namespaces of all projects (including future
ones).
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: k8s-admins-projects
spec:
  subjects:
    - kind: Group
      name: k8s-admins
  roleRef:
    kind: ClusterRole
    name: d8:project:admin
```

For `SuperAdmin`, replace the roles with `d8:system:superadmin` and `d8:project:superadmin`.

> **i**
>
> Every user namespace is a project: a namespace created directly becomes a project automatically. Therefore, the “system role + ClusterProjectRoleBinding” pair covers both the platform and all user namespaces. The pair does not cover the `default` namespace: it is a system namespace that belongs to the virtual project `default`, ClusterProjectRoleBinding does not extend to virtual projects, and no DP module runs in it.
>
> You cannot create a custom “all permissions” role (`apiGroups: ["*"], resources: ["*"], verbs: ["*"]`) — it is rejected by the built-in protection mechanisms. If you need unrestricted access outside the platform role model, use a ClusterRoleBinding to the built-in Kubernetes `cluster-admin` role — only someone who already has such permissions can assign it.

Additional resources

- Module `cni-cilium` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation
- Module `log-shipper` custom resources
- Module `multitenancy-manager` documentation

- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster admin access model

Deckhouse Platform (DP) supports the presence of multiple kubeconfig files on master nodes (this feature is implemented by the `control-plane-manager` module). Understanding their purpose is important for secure cluster administration.

Kubeconfig files on master nodes

The following kubeconfig files are located on the master nodes:

File	Identity	Purpose
`/etc/kubernetes/admin.conf`	`kubernetes-admin` (`kubeadm:cluster-admins` group)	Machine kubeconfig for kubeadm internals (join, renewal). The group is bound to the built-in `cluster-admin` role. With the `user-authz` module enabled, the group is additionally bound to two DP ClusterRoles
`/etc/kubernetes/super-admin.conf`	`kubernetes-super-admin` (`system:masters` group)	Break-glass emergency credential. Bypasses RBAC entirely. Restrict access to this file to trusted recovery scenarios
`/etc/kubernetes/controller-manager.conf`	`system:kube-controller-manager`	Used by kube-controller-manager
`/etc/kubernetes/scheduler.conf`	`system:kube-scheduler`	Used by kube-scheduler

RBAC-based admin access

Starting from Kubernetes 1.29, kubeadm generates `admin.conf` with the `kubeadm:cluster-admins` group instead of `system:masters`. This provides RBAC-controlled admin access that can be revoked by removing the `kubeadm:cluster-admins` ClusterRoleBinding(s).

DP binds the `kubeadm:cluster-admins` group to the built-in wildcard ClusterRole `cluster-admin`, regardless of whether the `user-authz` module is enabled. This is the same effective model as in a plain kubeadm cluster without extra RBAC.

If `user-authz` is **enabled**, two more ClusterRoleBindings bind the group to the following ClusterRoles:

- `d8:control-plane-manager:admin-kubeconfig-supplement` — rules for nodes, certificates, and other cluster machinery;
- `d8:user-authz:admin-kubeconfig` — managing AuthorizationRule and ClusterAuthorizationRule.

Recommended admin access

If the `user-authn` module is enabled, use a personalized OIDC-based kubeconfig obtained through the DP web interface. This provides individual accountability and audit trail.

If `user-authn` is disabled, administrators can explicitly use the admin kubeconfig on a master node:

```
d8 k --kubeconfig=/etc/kubernetes/admin.conf <command>
```

Root kubeconfig symlink

By default, the `control-plane-manager` module creates a symlink `/root/.kube/config` → `/etc/kubernetes/admin.conf` on master nodes, allowing root to run `d8 k` without specifying `--kubeconfig`.

When the `user-authz` module is enabled, you can disable this symlink by setting `rootKubeconfigSymlink: false` in the `control-plane-manager` module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

If `user-authz` is disabled, `control-plane-manager` does not apply this setting and keeps the default symlink behavior.

When the symlink is disabled (and `user-authz` is enabled), the symlink is removed when it pointed to `admin.conf`. Use personalized credentials or pass `--kubeconfig` explicitly.

Security hardening

The `control-plane-manager` module automatically restricts file permissions on `admin.conf` and `super-admin.conf` to `0600` (owner read/write only) during every reconciliation cycle. This prevents unauthorized users from reading these sensitive credentials.

Break-glass access

In emergency situations (RBAC misconfiguration, webhook failures), use `super-admin.conf`:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <command>
```

This credential bypasses all RBAC checks. Use it only as a last resort and restrict who can read the file.

Additional resources

- Module `control-plane-manager` documentation
- Module `user-authn` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Granting permissions to users and service accounts

To grant permissions in Deckhouse Platform (DP), you need to define a `subjects` block. Its format and contents depend on the resource type:

- In AuthorizationRule and ClusterAuthorizationRule resources (basic role model), as well as ProjectRoleBinding and ClusterProjectRoleBinding of the `multitenancy-manager` module (granular role model, project scope) — use the `subjects` block shown below.
- In standard Kubernetes RoleBinding and ClusterRoleBinding resources (granular role model, namespace/subsystem/whole-platform scope), the format differs: for `kind: User` and `kind: Group`, a `subjects` item additionally requires the `apiGroup: rbac.authorization.k8s.io` field (not needed for `kind: ServiceAccount`). For details, see the examples in Granting permissions using ClusterRoleBinding and RoleBinding below.

For users, it should be specified in the following format:

```
subjects:
- kind: User
  name: <user email>
```

⚠ If you are using the `user-authn` module and static users, make sure to specify the user's email in `subjects`, not the name of the User resource.

or

```
subjects:
- kind: Group
  name: <name of the group the user belongs to>
```

For a service account, the `subjects` block should be specified as follows:

```
subjects:
- kind: ServiceAccount
  name: <service account name>
  namespace: <namespace where the service account is created>
```

Granting permissions using AuthorizationRule and ClusterAuthorizationRule (basic role model)

When using the basic role model in DP, you can grant permissions to users via the AuthorizationRule and ClusterAuthorizationRule resources.

⚠ Support for the basic role model will be discontinued in future releases. For new roles, use the granular role model below — it can be used at the same time as the basic model, since the permissions they grant are summed up.

AuthorizationRule and ClusterAuthorizationRule objects can also be managed without writing YAML manifests, with the `d8 iam access` command: `d8 iam access grant` creates them, and `d8 iam access revoke` deletes them. For example, to grant the `Admin` access level to a user in the `dev` namespace, run:

```
d8 iam access grant user anton --access-level Admin -n dev
```

For more examples, see the `user-authz` module documentation.

Granting permissions to a user within a single namespace

To grant a user permissions within a single namespace, use the AuthorizationRule resource. It is applied within a single namespace.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: AuthorizationRule
metadata:
  name: dev-access
  namespace: dev-namespace
spec:
  subjects:
  - kind: User
    name: dev-user@example.com
  accessLevel: Admin
  portForwarding: true
```

Granting permissions to a user in all namespaces

To grant a user permissions across all namespaces (for example, to assign administrator permissions), use the ClusterAuthorizationRule resource. It is applied cluster-wide.

> **i** In multitenancy mode, a ClusterAuthorizationRule without `namespaceSelector` does not grant access to system namespaces: they require `namespaceSelector.matchAny: true` or the deprecated `allowAccessToSystemNamespaces: true`.

If needed, you can restrict the scope of permissions granted via ClusterAuthorizationRule to one or several namespaces. To do this, you can set the corresponding restrictions in the manifest. However, if possible, it is recommended that you use the AuthorizationRule resource for this purpose.

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin-access
spec:
  subjects:
  - kind: User
    name: dev-user@example.com
  # Available only with the enableMultiTenancy parameter of the user-authz module
  # turned on (BE, SE, SE+, EE, and CSE editions).
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
  accessLevel: Admin
  portForwarding: true
```

> **⚠** The `namespaceSelector` and `limitNamespaces` parameters only restrict access to objects in namespaces. A subject with the `ClusterAdmin` or `SuperAdmin` access level keeps its cluster-wide permissions, including creating ClusterRoleBinding and ClusterAuthorizationRule objects, so these parameters do not limit the permissions it can grant across the whole cluster. Grant these levels only to those who may be given access to the whole cluster.

Granting permissions using ClusterRoleBinding and RoleBinding (granular role model)

When using the granular role model in DP, you can grant permissions to users via the ClusterRoleBinding and RoleBinding resources — or, to grant access across all namespaces of a project at once, the ProjectRoleBinding and ClusterProjectRoleBinding resources of the `multitenancy-manager` module.

Roles of the granular model operate in one of four scopes — namespace, project, subsystem, or the whole platform — each with its own role name format and access levels. For details, see Role scopes in the granular role model documentation.

> **i** Access can also be granted without writing YAML manifests, with the access grant wizard in the Deckhouse web UI. The wizard walks you through the steps (who gets access → where → at which level), picks the binding kind itself (RoleBinding, ClusterRoleBinding, ProjectRoleBinding, or ClusterProjectRoleBinding), and lets you assemble a custom role from ready-made capabilities.

Assigning cluster administrator permissions (granular role model)

To assign cluster administrator permissions, use the system role `d8:system:manager` in a ClusterRoleBinding resource.

Example of assigning cluster administrator permissions to the user `jane.doe@example.com`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:system:manager
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user gets access to cluster-wide resources, to the configuration of DP modules, and to the namespaces of the enabled DP modules. The user gets no access to the namespaces of user applications.

The user will have the following permissions:

- View, modify, delete, and create cluster-wide Kubernetes resources and the cluster-wide resources of DP modules.
- Modify module configurations (view, edit, delete, and create ModuleConfig resources).
- View objects, logs, metrics, and events in the namespaces of DP modules, without access to Secrets.

The `d8:system:manager` role does not allow writing to the namespaces of DP modules, reading Secrets there, or running `kubectl exec`, `kubectl attach`, and `kubectl port-forward` on their Pods. These operations require the `d8:system:superadmin` role. For details, see Access of system and subsystem roles to the namespaces of the modules.

Assigning networking administrator permissions (granular role model)

To assign networking administrator permissions for managing the cluster's network subsystem, use the subsystem role `d8:subsystem:network:manager` in a ClusterRoleBinding resource.

Example of assigning networking administrator permissions to the user `jane.doe@example.com`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:subsystem:network:manager
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user's permissions will be limited to the following DP module namespaces from the `network` subsystem (the actual list depends on the modules enabled in the cluster):

- `d8-cni-cilium`
- `d8-cni-flannel`
- `d8-cni-simple-bridge`
- `d8-ingress-nginx`
- `d8-istio`
- `d8-keepalived`
- `d8-metallb`
- `d8-monitoring`
- `d8-network-gateway`
- `d8-openvpn`
- `d8-service-with-healthchecks`

- `d8-static-routing-manager`
- `d8-system`
- `kube-system`

The user will have the following permissions:

- View objects, logs, metrics, and events in the module namespaces from the `network` subsystem, without access to Secrets.
- View, modify, create, and delete the following resources of the modules from the `network` subsystem:
  - EgressGateway
  - EgressGatewayPolicy
  - FlowSchema
  - IngressClass
  - IngressIstioController
  - IngressNginxController
  - IPRuleSet
  - IstioFederation
  - IstioMulticluster
  - KeepalivedInstance
  - RoutingTable
- Modify module configurations (view, modify, create, and delete ModuleConfig resources) in the `network` subsystem.

  Modules the user will be able to manage:

  - `cilium-hubble`
  - `cni-cilium`
  - `cni-flannel`
  - `cni-simple-bridge`
  - `ingress-nginx`
  - `istio`
  - `keepalived`
  - `kube-dns`
  - `kube-proxy`
  - `metallb`
  - `monitoring-ping`
  - `network-gateway`
  - `network-policy-engine`
  - `node-local-dns`
  - `openvpn`
  - `sdn`
  - `service-with-healthchecks`
  - `static-routing-manager`

The `d8:subsystem:network:manager` role does not allow writing to the module namespaces, reading Secrets there, or running `kubectl exec`, `kubectl attach`, and `kubectl port-forward` on their Pods. These operations require the `d8:subsystem:network:superadmin` role. For details, see Access of system and subsystem roles to the namespaces of the modules.

Granting administrator permissions to a user within a namespace (granular role model)

To assign or restrict user permissions to specific namespaces, apply a namespace role with the corresponding access level in a RoleBinding resource.

For example, to allow a user to manage application resources in a namespace (without giving them access to DP module configurations), use the `d8:namespace:admin` role in a RoleBinding resource for the corresponding namespace. To grant the same access across all namespaces of a project at once, use a ProjectRoleBinding with the equivalent `d8:project:admin` role instead.

Example of granting application developer `app-developer` permissions within the `myapp` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:namespace:admin
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user's permissions will be limited to the following within the `myapp` namespace:

- View, modify, create, and delete Kubernetes resources. Example resources:
  - Certificate
  - CertificateRequest
  - ConfigMap
  - ControllerRevision
  - CronJob
  - DaemonSet
  - Deployment
  - Event
  - HorizontalPodAutoscaler
  - Ingress
  - Issuer
  - Job
  - Lease
  - LimitRange
  - NetworkPolicy
  - PersistentVolumeClaim
  - Pod
  - PodDisruptionBudget
  - ReplicaSet
  - ReplicationController
  - ResourceQuota
  - Role
  - RoleBinding
  - Secret
  - Service
  - ServiceAccount
  - StatefulSet
  - VerticalPodAutoscaler
  - VolumeSnapshot
- View, modify, create, and delete the following DP module resources:
  - DexAuthenticator
  - DexClient
  - PodLoggingConfig
- Run the following commands on Pods and Services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

Additional resources

- Module `multitenancy-manager` custom resources
- Module `user-authn` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Checking user permissions

Quick check of a single permission

The quickest way to check a single permission is the `d8 k auth can-i` command:

```
d8 k auth can-i --as=user@example.com get pods -n other-namespace
```

To see the full set of actions that a user, group, or ServiceAccount is allowed to perform — not just a single action on a single resource — use the SubjectAccessReport resource described below.

Checking whether a user has access

To check whether a user has access permissions, run the following command, which specifies:

- `resourceAttributes` (as in RBAC) — what access is checked for;
- `user` — the user name;
- `groups` — the user groups.

> When used together with the `user-authn` module, the user name and groups can be found in the Dex logs with the `d8 k -n d8-user-authn logs -l app=dex` command. They appear in the logs only after the user signs in.

```
cat  <<EOF | 2>&1 d8 k create --raw
/apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "dev",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "user@example.com",
    "groups": [
      "developers"
    ]
  }
}
EOF
```

The response will show whether access is allowed and which role grants it.

Example response if the user has access permissions:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by RoleBinding \"<BINDING_NAME>/dev\" of ClusterRole
\"<ROLE_NAME>\" to User \"user@example.com\""
}
```

Example response if the user does not have access permissions:

```
{
  "allowed": false
}
```

In **multitenancy** mode, the namespace restrictions of the basic role model (ClusterAuthorizationRule/AuthorizationRule) are enforced by the authorization webhook. The answer of the API server already includes the decision of the webhook. To see the answer of the webhook alone, run the following command as root on a master node:

```
cat <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-
config.yaml create --raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "dev",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "user@example.com",
    "groups": [
      "developers"
    ]
  }
}
EOF
```

Example response if the user does not have access permissions:

```
{
  "allowed": false
}
```

A response with `"allowed": false` means the webhook is not blocking the request. If the webhook blocks the request, you get, for example, the following message:

```
{
  "allowed": false,
  "denied": true,
  "reason": "either you have no access to the namespace or the namespace does not
exist"
}
```

Getting a full permission report

With multitenancy mode enabled (`enableMultiTenancy`), the `SubjectAccessReport` resource of the `user-authz` module gives a ready-made report of everything a subject is allowed to do, instead of checking one verb on one resource at a time: which roles are granted through which bindings, which actions on which resources are allowed cluster-wide and in every namespace, and where each permission comes from.

Example of getting a report:

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: SubjectAccessReport
metadata:
  name: what-can-jane-do
spec:
  subject:
    kind: User
    name: jane@example.com
EOF
```

The `spec.subject.kind` field accepts `User`, `Group`, and `ServiceAccount` (for the latter, `spec.subject.namespace` is required). If `spec.subject` is omitted, the report is generated for the caller.

A report about oneself is available to every authenticated user and requires no extra permissions. Building a report about **another** subject requires the `d8:user-authz:subject-access-checker` cluster role, which is not bound to anyone by default.

There is also a reverse query — the `WhoCan` resource — which answers “who can perform action X on resource Y?” and returns the list of users, groups, and ServiceAccounts. Example:

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: WhoCan
metadata:
  name: who-can-create-networkpolicies
spec:
  resourceAttributes:
    namespace: my-namespace
    verb: create
    group: networking.k8s.io
    resource: networkpolicies
EOF
```

The answer is returned in the `status` field (it contains the `users`, `groups`, and `serviceAccounts` fields) right in the command output. The object is not stored anywhere.

The right to create `WhoCan` queries is granted by the `d8:user-authz:who-can-checker` cluster role. It is not bound to anyone by default: the result of a query reveals the access subjects in all namespaces, so grant it only to trusted administrators through a ClusterRoleBinding.

The managers of the `iam` and `security` subsystems can build reports about other subjects and create `WhoCan` queries as well. This right comes with the `d8:subsystem-capability:iam:inspect_access` capability of the `d8:subsystem:iam:manager` and `d8:subsystem:security:manager` roles, so the roles above them have it too (`d8:subsystem:iam:superadmin`, `d8:subsystem:security:superadmin`, `d8:system:manager`, and `d8:system:superadmin`). For details on both resources, see the `user-authz` module FAQ.

Additional resources

- Module `user-authn` documentation
- Module `user-authz` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Access for CI/CD

The following methods are available for authenticating CI/CD pipelines in the Kubernetes API:

- ServiceAccount — Kubernetes ServiceAccount token.
- Basic Auth — username and password via IdP.
- Token Exchange — exchange IdP token for Dex token.

ServiceAccount

The ServiceAccount token is used directly for API authentication. No external IdP is required. For ServiceAccount, you can create long-lived tokens (without TTL, with the possibility of revocation or rotation) and short-lived tokens.

When multiple pipelines share a single ServiceAccount, audit logs will not contain information about specific pipelines.

Prerequisites

To configure token-based authentication for ServiceAccount, the following requirements must be met:

- Cluster access with permissions to create ServiceAccounts and Secrets.
- For external access: apiserver.publishAPI of the `control-plane-manager` module, or direct API access via VPN.

Create ServiceAccount and a long-lived token

To create a ServiceAccount and token, use the command:

```
d8 k create ns ci-deploy || true

cat <<EOF | d8 k apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: ci-deploy
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: ci-deploy
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

> **i** The `kubernetes.io/service-account-token` Secret type is a legacy approach. The recommended method is the TokenRequest API (`d8 k create token ...`).

Grant permissions

For details on granting permissions, see Granting permissions to users and service accounts .

Assign the necessary permissions to the ServiceAccount.

For the basic role model, use ClusterAuthorizationRule:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-access
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci-deploy
  accessLevel: Admin
  portForwarding: true
EOF
```

Available levels: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`.

For the granular role model, grant the `d8:project:admin` role in all projects with a ClusterProjectRoleBinding. To grant access to a single namespace only, create a RoleBinding to the `d8:namespace:admin` role in it instead:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: gitlab-runner-access
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci-deploy
  roleRef:
    kind: ClusterRole
    name: d8:project:admin
EOF
```

Get API URL and CA certificate

When using publishAPI:

- When the API server is published via Ingress
- When the API server is published via Gateway API (`alb`)

When the API server is published via Ingress, use the following commands:

```
API_HOST=$(d8 k -n kube-system get ingress kubernetes-api -o
jsonpath='{.spec.rules[0].host}')
echo "API endpoint: https://${API_HOST}"
```

When the API server is published via Gateway API (the `alb` module), use the following commands:

```
API_HOST=$(d8 k -n kube-system get httproute kubernetes-api -o
jsonpath='{.spec.hostnames[0]}')
echo "API endpoint: https://${API_HOST}"
```

> **i** If the API certificate is signed by a public CA (Let’s Encrypt), the `--certificate-authority` parameter is not required.

For a private CA (self-signed, the default mode — shared between both publication methods):

```
d8 k -n kube-system get secret kubernetes-api-ca-key-pair -o jsonpath='{.data.tls\.crt}' |
base64 -d > /tmp/ca.crt
```

Create kubeconfig

Example of generating kubeconfig for a previously created ServiceAccount:

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config

# For public CA, remove --certificate-authority and --embed-certs
d8 k config set-cluster $CLUSTER_NAME \
  --server=https://${API_HOST} \
  --certificate-authority=/tmp/ca.crt \
  --embed-certs=true \
  --kubeconfig=$FILE_NAME

d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n ci-deploy get secret gitlab-runner-deploy-token -o jsonpath='{.data.token}'
| base64 -d) \
  --kubeconfig=$FILE_NAME

d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME

d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

Short-lived tokens (TokenRequest)

The TokenRequest API allows creating tokens with limited lifetime. When creating such a token, specify the name of the ServiceAccount that needs to be granted access in the command (in the example — `gitlab-runner-deploy`):

```
d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h
```

⚠ The `d8 k create token` command itself requires access to the Kubernetes API. If the CI job has no initial cluster access, it cannot execute TokenRequest from within the job.

Usage patterns in CI

Bootstrap token with minimal permissions. CI stores a long-lived kubeconfig/token that only has permission to create TokenRequest for a specific ServiceAccount. At the start of the job, the bootstrap token issues a short-lived token used for deployment.

External token broker. An operator, secret manager (Vault, External Secrets), or external service periodically issues short-lived tokens and places them in CI variables or secrets. The CI job only consumes the ready token.

ℹ The bootstrap token should have minimal permissions: only `create` on the `serviceaccounts/token` resource for a specific ServiceAccount/namespace. Do not use cluster-admin. Limit the TTL of issued tokens and promptly revoke bootstrap access if compromised.

Usage without kubeconfig file:

```
export KUBE_SERVER="https://${API_HOST}"
export KUBE_TOKEN=$(d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h)
d8 k --server=$KUBE_SERVER --token=$KUBE_TOKEN get ns
```

Basic Auth

Authentication via username and password through IdP (LDAP, OIDC).

⚠ The password is sent to DP and validated through basic-auth-proxy/Dex.

⚠ Only one DexProvider in the cluster can have `enableBasicAuth: true`.

Prerequisites

The following requirements must be met to configure Basic Auth:

- apiserver.publishAPI of the `control-plane-manager` module enabled.
- DexProvider configured for IdP.

Enable

Add `enableBasicAuth: true` to DexProvider. For DexProvider configuration examples, see user-authn module documentation.

Get API endpoint

To get the API endpoint, use the command:

```
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o
jsonpath='{.spec.rules[0].host}')
echo "https://${API_HOST}"
```

Verification

To check access, use the command:

```
curl -q -u "$K8S_USER:$K8S_PASSWORD" "https://${API_HOST}/version"
```

Possible errors:

- `401` — invalid credentials or Basic Auth not enabled.
- `403` — authentication succeeded but RBAC denies access.

Configure kubeconfig

Example of kubeconfig configuration for Basic Auth:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://<API_HOST>
users:
- name: basic-auth-user
  user:
    username: "<USERNAME>"
    password: "<PASSWORD>"
contexts:
- name: default
  context:
    cluster: my-cluster
    user: basic-auth-user
current-context: default
```

Use in GitLab CI

Example of using Basic Auth in GitLab CI:

```
deploy:
  script:
    - d8 k --server="$K8S_SERVER" --username="$K8S_USER" --password="$K8S_PASSWORD"
get ns
```

Variables `K8S_SERVER`, `K8S_USER`, `K8S_PASSWORD` are set in the project CI/CD settings.

Token Exchange

The pipeline gets a token from its CI system, exchanges it at Dex for a token with `aud=kubernetes`, and uses it to access the API.

> **i** Recommended for GitLab CI and GitHub Actions.

DP/Dex receives no password. Dex trusts the CI system as a TokenExchangeProvider. It checks the signature, the issuer, the expiry and the audience of the token and the required claims, and names the pipeline in the cluster by a claim of the token.

Prerequisites

The following requirements must be met to configure Token Exchange:

- apiserver.publishAPI of the `control-plane-manager` module enabled.
- The CI system issues signed JWT ID tokens (GitLab CI `id_tokens`, GitHub Actions OIDC) and publishes its discovery document, or you have its public keys.

Create DexClient

To create DexClient, use the command:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: ci-token-exchange
  namespace: d8-user-authn
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec: {}
EOF
```

The annotation `dexclient.deckhouse.io/allow-access-to-kubernetes` allows the client to request tokens with `aud=kubernetes`.

> ⚠ This grants cluster-wide access to the Kubernetes API. For this reason, adding the annotation or changing it to `"true"` requires the `create` permission on the `dexclients/allow-access-to-kubernetes` subresource of the `deckhouse.io` API group in the DexClient namespace. In the basic role model, this permission is granted at the `ClusterAdmin` level and above. In the granular model, it is granted by the `d8:subsystem:iam:manager` role and the roles that include it, such as `d8:system:manager`. A namespace-level role that permits creating a DexClient is not enough: the admission policy rejects the request with a message naming the annotation.
>
> The addition of the annotation is restricted regardless of the specified value, including `"false"`. This is required to ensure compatibility with previous DP versions where access has been granted when the annotation was present, regardless of its value.
>
> This restriction is not applicable to an object, which already has the annotation assigned. Deletion and recreation of DexClient is treated as another addition of the annotation, so a GitOps controller that recreates the object instead of patching it requires the same `create` permission on the `dexclients/allow-access-to-kubernetes` subresource.

Get client secret:

```
d8 k -n d8-user-authn get secret dex-client-ci-token-exchange -o jsonpath='{.data.clientSecret}' | base64 -d
```

client_id format: `dex-client-ci-token-exchange@d8-user-authn`.

Create TokenExchangeProvider

TokenExchangeProvider tells Dex which tokens to accept and which DexClient may exchange them. Example for GitLab CI:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1alpha1
kind: TokenExchangeProvider
metadata:
  name: gitlab-ci
spec:
  issuer: https://gitlab.example.com
  keys:
    discovery: {}
  audiences:
  - exact: https://<DEX_HOST>/
  claimValidation:
  - claim: namespace_path
    equals: platform
  - claim: ref_protected
    equals: "true"
  identity:
    username:
      claim: sub
      prefix: "gitlab-ci:"
  clients:
  - dexClient:
      name: ci-token-exchange
      namespace: d8-user-authn
EOF
```

Parameters:

- `issuer`: URL of the GitLab instance, as in the `iss` claim of its ID tokens.
- `audiences`: Value of `aud` that the job sets in `id_tokens`. A token issued for another service is refused.
- `claimValidation`: Claims that a token must carry. In the example, only protected branches of projects in the `platform` group are accepted.
- `identity.username`: User name in the cluster, for example `gitlab-ci:project_path:platform/app:ref_type:branch:ref:main` for the `main` branch of `platform/app`.

Get the connector ID for the exchange request:

```
CONNECTOR_ID=$(d8 k get tokenexchangeprovider gitlab-ci -o jsonpath='{.status.connectorID}')
```

Store it as a CI variable. The ID changes if the object is deleted and created again, or if the provider gets its connector back after a `PrefixConflict`. In these cases, update the variable, otherwise Dex answers with 400 `Requested connector does not exist`.

Get Dex and API URLs

To get the Dex URL and API, use the following commands:

```
DEX_HOST=$(d8 k -n d8-user-authn get ingress dex -o jsonpath='{.spec.rules[0].host}')
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o
jsonpath='{.spec.rules[0].host}')
```

Grant RBAC

DP configures kube-apiserver to validate Dex tokens. The user name is the one TokenExchangeProvider sets in `identity.email` if it is set, and in `identity.username` otherwise; groups are taken from `identity.groups`.

For details on granting permissions, see Granting permissions to users and service accounts.

For the basic role model, use ClusterAuthorizationRule:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: ci-deployer
spec:
  subjects:
  - kind: User
    name: "gitlab-ci:project_path:platform/app:ref_type:branch:ref:main"
  accessLevel: Admin
EOF
```

For the granular role model, grant the `d8:project:admin` role in all projects with a ClusterProjectRoleBinding. To grant access to a single namespace only, create a RoleBinding to the `d8:namespace:admin` role in it instead:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: ci-deployer
spec:
  subjects:
  - kind: User
    name: "gitlab-ci:project_path:platform/app:ref_type:branch:ref:main"
  roleRef:
    kind: ClusterRole
    name: d8:project:admin
EOF
```

Token exchange request

Example of a token exchange request:

```
RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
  -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
  -d "subject_token=${CI_TOKEN}" \
  -d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
  -d "connector_id=${CONNECTOR_ID}" \
  -d "scope=openid email groups federated:id audience:server:client_id:kubernetes" \
  -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")

DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
```

Parameters:

- `subject_token`: ID token of the CI system. Access tokens are not accepted.
- `connector_id`: `status.connectorID` of TokenExchangeProvider.
- `scope`: Must include the following values:
  - `openid`: Required for an ID token.
  - `email`: kube-apiserver takes the user name from the `email` claim. Without it, the API answers 401.
  - `audience:server:client_id:kubernetes`: The token is issued for kube-apiserver.
  - `groups`: Required if RBAC grants rights to groups.
  - `federated:id`: Shows the TokenExchangeProvider of a token in `d8 k auth whoami` (the `user-authn.deckhouse.io/dex-provider` extra).

The Dex token lives as long as `idTokenTTL` of the `user-authn` module (10 minutes by default). Request a new one for a longer job.

GitLab CI

Example of a token request for GitLab CI:

```
deploy:
  id_tokens:
    GITLAB_OIDC_TOKEN:
      aud: https://<DEX_HOST>/
  script:
    - |
      RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
        -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
        -H "Content-Type: application/x-www-form-urlencoded" \
        -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
        -d "subject_token=${GITLAB_OIDC_TOKEN}" \
        -d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
        -d "connector_id=${CONNECTOR_ID}" \
        -d "scope=openid email groups federated:id
audience:server:client_id:kubernetes" \
        -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")
      DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
      d8 k --server="${K8S_SERVER}" --token="${DEX_TOKEN}" auth whoami
```

GitHub Actions

For GitHub Actions, set the following in TokenExchangeProvider:

- `issuer`: `https://token.actions.githubusercontent.com`.
- `audiences`: Value that the workflow passes as `audience`.
- `claimValidation`: `repository_owner_id` and, if needed, `ref`. GitHub recommends checking the IDs, which never change. The name of a renamed or deleted organization can be taken by someone else.
- `identity.username`: `claim: sub` with a prefix, for example `github:`. The user name becomes `github:repo:my-org/app:ref:refs/heads/main`.

To request the token, add the `id-token: write` permission to the workflow:

```
permissions:
  id-token: write
steps:
  - run: |
      CI_TOKEN=$(curl -sS -H "Authorization: bearer
${ACTIONS_ID_TOKEN_REQUEST_TOKEN}" \
        "${ACTIONS_ID_TOKEN_REQUEST_URL}&audience=https://<DEX_HOST>/" | jq -r
'.value')
```

Exchange `CI_TOKEN` as in the token exchange request.

Migration from DexProvider

Earlier, a token was exchanged through a DexProvider of OIDC type, and the exchange request named it in `connector id`. This method is deprecated and off by default. A DexProvider exchanges tokens only with `spec.tokenExchange.enabled: true`. The module set this value during the upgrade in clusters that had a DexClient with the `dexclient.deckhouse.io/allow-access-to-kubernetes` annotation. The parameter can only be turned off. It will be removed after the next LTS release, together with this method. While a cluster exchanges tokens through a DexProvider, the `D8UserAuthnDexProviderTokenExchangeDeprecated` alert fires.

To migrate, follow these steps:

1. Create a TokenExchangeProvider with the `issuer` of the DexProvider and the audience your CI tokens carry.
2. Set `identity.username` to the claim the DexProvider took the user name from, for example `claim: user_email` for GitLab CI, with a prefix such as `gitlab-ci:`; give the groups a prefix too. The prefix is required. It sets the subjects of the provider apart from people and from the subjects of other providers.
3. Keep the checks of the DexProvider:
   - A DexProvider accepted only a token with `email_verified: true`, unless

`insecureSkipEmailVerified` was set. If the user name comes from an email, add `claimValidation: [{claim: email verified, equals: "true"}]`.
    - Move `allowedEmails`, `allowedEmailDomains` and `allowedGroups` to `allowedIdentities`. Dex applies them to the names with the prefixes added, such as `gitlab-ci:deployer@example.com`. Add the prefixes of `identity` to `allowedEmails` and `allowedGroups`. Move `allowedEmailDomains` as they are, because the domain is the part after the last `@`.
    - `allowedGroups` needs `identity.groups`. Without groups, every exchange is refused.
4. Before the switch, add the prefixed subjects to the ClusterAuthorizationRule, AuthorizationRule and RoleBinding objects that grant rights to the old names, for example `gitlab-ci:deployer@example.com` next to `deployer@example.com`.
5. In the exchange request, replace `connector_id` with `status.connectorID` of the new object.
6. After the switch, remove the old subjects from these objects.
7. Turn off token exchange through the DexProvider by setting `spec.tokenExchange.enabled: false`. It cannot be turned on again.

A TokenExchangeProvider accepts JWT ID tokens only. A pipeline that exchanged an access token, for example one of Keycloak, has to request an ID token of the issuer instead.

Diagnostics

> **i** Use `d8 k auth whoami` to verify which identity is being used.

`--token` flag may not work — kubeconfig with `auth-provider`, `exec`, or `tokenFile` overrides the token from the flag. Use `-v=8` for debugging:

```
d8 k --server="https://${API_HOST}" --token="${DEX_TOKEN}" auth whoami -v=8
```

The output shows which Authorization header is sent. For reliable token usage, use a minimal kubeconfig with only `server` and `token`, without `auth-provider`/`exec`.

TokenExchangeProvider is not ready — check the `Ready` condition of the object:

```
d8 k get tokenexchangeprovider gitlab-ci -o jsonpath='{.status.conditions}'
```

Dex 400 — one of the following:

- An unknown `subject_token_type`, or a wrong `connector_id` (`Requested connector does not exist`).
- The DexClient is not listed in `spec.clients` (`Connector not allowed for this client.`).
- `invalid_scope`: The DexClient has no `dexclient.deckhouse.io/allow-access-to-kubernetes` annotation, so it may not request `audience:server:client_id:kubernetes`. If the annotation is gone, check whether the admission-controller rejected the request that was supposed to add it, or whether a GitOps controller recreated the object without it. Adding the annotation requires the `create` permission on the `dexclients/allow-access-to-kubernetes` subresource of the `deckhouse.io` API group in the DexClient namespace.
- `unauthorized_client`: The client is a kubeconfig client or `kubernetes`, which may not exchange tokens.

Dex 401 — invalid client credentials, or the token is refused: an access token instead of an ID token, wrong signature or issuer, a `kid` of no pinned key, expired or valid for longer than `subjectTokenMaxLifetime`, another audience, a required claim does not match, or the names with the prefixes added are not in `allowedIdentities` (`User is not allowed.`). The Dex log names the reason.

API 401 — token validation failed. Check:

- `scope` contains `email` and `audience:server:client_id:kubernetes`.
- Time synchronization between CI runner and cluster.

API 403 — authentication succeeded but RBAC does not allow access for the user or group from the token, or TokenExchangeProvider was deleted or lost its connector (`PrefixConflict`). In this case, every token issued through it is refused if the `enableMultiTenancy` parameter of the `user-authz` module is enabled and the cluster runs Kubernetes 1.34 or later.

Decode token to check claims:

```
echo "${DEX_TOKEN}" | cut -d. -f2 | base64 -d 2>/dev/null | jq .
```

Additional resources

- Module `alb` documentation
- Module `control-plane-manager` parameters
- Module `multitenancy-manager` custom resources
- Module `user-authn` documentation
- Module `user-authz` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Resource listing with access-based filtering

In Kubernetes, the `kubectl get <resource> -A` command requires cluster-wide permissions for the `list` operation. Otherwise, kube-apiserver returns the `403 Forbidden` error, even if the user has access to dozens of namespaces individually.

Deckhouse Platform (DP) extends kube-apiserver so that a user without cluster-wide permissions can explicitly request a response filtered down to what they can actually access, instead of a refusal.

The mechanism is strictly opt-in. For a client who does not request filtering, the behavior remains identical to the standard one used in Kubernetes.

Using via Deckhouse CLI

In DP, you can use the Deckhouse CLI tool to work with the filtered listing mechanism.

Resource filtering scopes

Using the `--scope` flag, you can define namespaces the resources will be returned from.

Value	What is returned
`accessible`	Objects from every namespace where the user's RBAC allows the given operation on the given resource
`projects`	Objects from namespaces belonging to any project, that is, namespaces with a non-empty value of the `projects.deckhouse.io/project` label
`project:<name>`	Objects from the namespaces of one specific project. The name must be a valid project name
`system`	Objects from system namespaces only: `default` and namespaces whose names start with `d8-` or `kube-`. This is a fixed list of namespaces defined by their names and independent of projects. Other namespaces that neither belong to a project nor match the list of system namespaces are not returned when setting either `system` or `projects` scope

The `projects` and `project:<name>` scopes require the `multitenancy-manager` module. While the module is disabled, requests with these scopes are rejected with the `400 Bad Request` error.

Examples

Use the `--scope` flag in the `d8 k get` command together with the `-A` flag (or `--all-namespaces`).

Obtaining objects from all namespaces available to the user:

```
d8 k get pods -A --scope=accessible
```

Obtaining objects from project namespaces only:

```
d8 k get pods -A --scope=projects
```

Obtaining objects from a specified project:

```
d8 k get pods -A --scope=project:my-project
```

The mechanism is applicable to custom resources as well:

```
d8 k get configauditreports -A --scope=accessible
```

The filtering can be applied to watch requests as well:

```
d8 k get pods -A --scope=accessible -w
```

Details:

- Deckhouse CLI rejects `--scope` without the `-A` flag and together with the `-n` (`--namespace`) flag: in a single namespace, the flag could only narrow that namespace's listing, usually to an empty one, instead of returning the `403 Forbidden` error. The HTTP API accepts the headers for a single namespace as well.
- If the `d8 k get <resource> -A` command without `--scope` fails with `403 Forbidden`, Deckhouse CLI suggests suitable `--scope` variants.
- Shell completion works for the `--scope` flag values, including the names of existing projects (`--scope=project:<TAB>`).

Example of an interface query for where the user may perform actions

With the multitenancy mode enabled (`enableMultiTenancy`), you can use the SelfScopedAccessReview resource to request information about the namespaces in which the user is allowed to perform actions. The query gives the user who creates it information on where actions are allowed for them within a filtering scope. One query replaces an access check per namespace. The namespaces are counted by the rule that filters a list sent with the same scope, so a section built on the answer opens exactly when its list has objects to return.

Any authenticated user can create the resource, and it answers only about the caller. A query with the `spec.user`, `spec.groups`, `spec.uid` or `spec.extra` fields is refused.

Example query from a Deckhouse web interface page that shows deployments of all projects, offers to create one and shows pod logs:

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: SelfScopedAccessReview
spec:
  scope: projects
  requests:
  - group: apps
    resource: deployments
    verbs: [list, watch]
  - group: apps
    resource: deployments
    verbs: [create]
    includeNamespaces: true
  - group: ""
    resource: pods
    subresource: log
    verbs: [get]
    includeNamespaces: true
EOF
```

Example answer for a user who administers the `shop-dev` and `shop-stage` namespaces of a project and has read access without pod logs in `shop-prod`:

```
status:
  results:
  - allowed: true
    cluster: false
    count: 3
  - allowed: true
    cluster: false
    count: 2
    namespaces: [shop-dev, shop-stage]
  - allowed: true
    cluster: false
    count: 2
    namespaces: [shop-dev, shop-stage]
```

Example answer for a cluster administrator. The namespaces are neither counted nor listed, even with `includeNamespaces: true`:

```
status:
  results:
  - allowed: true
    cluster: true
  - allowed: true
    cluster: true
  - allowed: true
    cluster: true
```

Example answer for a user who can only read deployments in the `billing-prod` namespace of another project:

```
status:
  results:
  - allowed: true
    cluster: false
    count: 1
  - allowed: false
    cluster: false
    count: 0
    namespaces: []
  - allowed: false
    cluster: false
    count: 0
    namespaces: []
```

For details on the SelfScopedAccessReview resource (query), including how it works, the query and answer fields, errors and usage recommendations, refer to the `user-authz` module documentation.

Using from your own code (HTTP API)

The CLI is just a transport mechanism. The filtering is enabled by two HTTP request headers sent to kube-apiserver. To use the mechanism from you own service, a script, or any other client, set the headers yourself.

Header	Value	When required
`X-Deckhouse-Scope`	`accessible` | `projects` | `system` | `project`	Always (enables the mechanism)
`X-Deckhouse-Project`	Project name	Only together with `X-Deckhouse-Scope: project`

The headers only affect read operations (`list`, `get`, `watch`) of top-level namespaced resources without a subresource. They apply both on a cluster-scoped URL (a path without `namespaces/<name>`, which is produced when the `-A` flag is used) and on the URL of one namespace (`namespaces/<name>/...`). On the URL of one namespace, objects are returned only if that namespace

is in the resulting set of namespaces: while it is outside the set, `list` and `watch` return no objects, and `get` returns `404 Not Found`.

Mutating requests and requests to subresources ignore the headers, whatever their value. For reads of cluster-scoped resources, a valid header has no effect, while an invalid value is rejected with the `400 Bad Request` error, as for any other read.

Example of listing the pods available to a service account:

```
curl -H "Authorization: Bearer $TOKEN" \
     -H "X-Deckhouse-Scope: accessible" \
     https://<kube-apiserver>/api/v1/pods
```

Example of listing the Deployment objects of one project:

```
curl -H "Authorization: Bearer $TOKEN" \
     -H "X-Deckhouse-Scope: project" \
     -H "X-Deckhouse-Project: my-project" \
     https://<kube-apiserver>/apis/apps/v1/deployments
```

Response semantics:

Situation	Response
No headers	Standard Kubernetes behavior (the `403 Forbidden` error for a user without permissions)
Header present but RBAC denied the request	`200 OK` with a list of objects from accessible namespaces (the list can be empty)
Header present, and the user has cluster-wide permissions	`200 OK`. The filtering is applied by namespace classification only (for example, `system` keeps only `default`, `d8-*`, and `kube-*` namespaces)
Invalid header value: an unknown scope, `X-Deckhouse-Project` without `X-Deckhouse-Scope: project`, or a project name that is not a valid name	`400 Bad Request` for every user, before the permissions are checked
The `projects` or `project` scope while the `multitenancy-manager` module is disabled	`400 Bad Request`
`get` of an object outside the resulting set of namespaces	`404 Not Found`, as for an object that does not exist
RBAC denied the request, and `permission-browser` cannot answer now: it does not answer in time, is overloaded, or is unreachable	`503 Service Unavailable` with the `Retry-After` header; no objects are returned
RBAC denied the request, and the access boundary cannot be computed for another reason	`403 Forbidden` (fail-closed)

When using client-go, add the headers via `rest.Config.WrapTransport`.

How it works

1. **Kube-apiserver authorization filter.**

   If RBAC denied the request, the filter — instead of returning `403 Forbidden` immediately — checks three conditions:

   - The read operation (`list`, `get`, or `watch`) is underway.
   - The request is addressed to the main resource and not the subresource.
   - A filtering decorator is registered for this resource and the header is present.

   Only when all three conditions are met, the request is passed on, with a mark of filtering required at the storage layer.

2. **The storage layer** computes the final namespace set as the intersection of two independent sets:

   - **Classification**: Which namespaces match the `--scope` value. Determined by the `projects.deckhouse.io/project` label that the `multitenancy-manager` module assigns to

project namespaces. Resolved by a direct request to kube-apiserver and does not depend on the user.
- **Access boundary** (with RBAC floor in the mechanism's code): Which namespaces are actually available to the user for *this resource and this operation*. Computed with one `SelfScopedAccessReview` request to the `authorization.deckhouse.io` API (the `user-authz` module) for the exact resource, subresource, and verb, the same answer the web interface uses to decide what to show. Applied whenever RBAC denies the request. The boundary also includes the namespaces where a RoleBinding grants permissions to every authenticated user (the `system:authenticated` group), although the namespace list does not show them.

Only objects from namespaces in the final set make it into the response.

3. **Watch** (`-w`): Returns a filtered stream, and if the set of accessible namespaces changes "on the fly" (for example, permissions granted or revoked, or a namespace was moved to another project), it synthesizes `ADDED`/`DELETED` events for the affected objects without the client reconnecting.

   The set is refreshed every second. A synthesized `DELETED` event carries the last `resourceVersion` of the stream. If the set cannot be refreshed (for example, `permission-browser` does not respond), the stream keeps the previous set for up to 60 seconds, then sends an error and closes. The client reconnects and gets `503 Service Unavailable` while `permission-browser` cannot answer, or `403 Forbidden` while the set cannot be computed for another reason. If the `multitenancy-manager` module is disabled while a watch with the `projects` or `project` scope is open, the stream sends a `400 Bad Request` error and closes at once. If `permission-browser` answers that the user may now perform the request in the whole cluster, the stream sends `410 Gone` and closes, and the client lists again.

The mechanism supports all built-in namespaced Kubernetes resources and namespaced resources defined via CRD. Resources of aggregated APIs take part only when their `APIService` opts in, see below.

Aggregated APIs

A resource served by an aggregated API server (an `APIService` with `spec.service`) takes part in the mechanism only when its `APIService` lists it in the `scope.deckhouse.io/resources` annotation, a comma-separated list of resource names:

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1.example.deckhouse.io
  annotations:
    scope.deckhouse.io/resources: widgets,gadgets
```

Without the annotation a read with a scope that RBAC denied gets `403 Forbidden`, as before. Setting the annotation is a cluster administrator decision: the aggregated API server then receives scoped reads and must filter its answers itself.

How a scoped read of an opted-in resource works:

- kube-apiserver computes the resulting set of namespaces as for a built-in resource and sends it to the aggregated API server in the `X-Deckhouse-Scope-Namespaces` header, with `X-Deckhouse-Scope-Mode: filtered` (RBAC denied the read) or `narrowed` (RBAC allowed it, the set only narrows the answer). Copies of these headers sent by the client are removed. The aggregated API server must trust them only from a request authenticated with the front proxy client certificate of kube-apiserver.
- `get` of an object in a namespace outside the set is answered by kube-apiserver with `404 Not Found`, without a request to the aggregated API server.
- A set that does not fit the request header limit of the aggregated API server (1 MiB) is answered with `400 Bad Request`; use a narrower scope.
- Local `APIService` objects (built-in groups and CRDs) and the `authorization.deckhouse.io` API ignore the annotation.
- `permission-browser` follows the annotation through a watch on `APIService` objects, so adding or removing it takes effect within seconds.

A watch of an opted-in resource gets the set once, when it opens. kube-apiserver caps its `timeoutSeconds` at 5 minutes, recomputes the set every second while the stream is open, and ends the stream when the set changes, when the annotation is removed, or when the set cannot be recomputed for 60 seconds. It does not synthesize `ADDED` and `DELETED` events as for a built-in resource. Over HTTP, a JSON or protobuf stream ends with a `410 Gone` error event, so a client-go informer lists again and gets objects of the new set. A stream in another encoding (for example, CBOR),

a compressed stream, a stream with an event larger than 16 MiB, or one kube-apiserver cannot split into events is closed without that event. A WebSocket watch is closed without that event: a client that watches over WebSocket must list again after the connection closes.

Security model

- **Standard Kubernetes behavior without headers.** Controllers and operators relying on `403 Forbidden` as a signal continue normal operation.
- **Headers do not widen access permissions.** The access boundary is computed on the server based on the identity of a user who made the request. Even if a client inserts the headers manually they can't access objects out of what their RBAC allows. The check is made for each pair (`resource`, `verb`). Therefore, for example, a user with the `get secrets` permission in a namespace can't access pods in it.
- **Using the fail-closed model.** If the `authorization.deckhouse.io` API is not served (for example, if the `user-authz` module is disabled, or `permission-browser` does not serve `SelfScopedAccessReview` yet during an update), the access boundary cannot be computed — a denied user gets the standard `403 Forbidden` response. The same response is returned when `permission-browser` allows the operation in the whole cluster while kube-apiserver has denied it: the mechanism never answers with all namespaces. If `permission-browser` cannot answer now (it does not answer in time, answers `429 Too Many Requests` or a `5xx` error, or is unreachable), a scoped request gets `503 Service Unavailable` with the `Retry-After` header instead: the access is unknown rather than denied, and the client may retry. Neither response returns any objects.
- **Indistinguishable refusals.** Requests like `project:<someone else's>` and `project:<nonexistent>` result in identical empty responses and project existence is not disclosed.

Special case: namespaces

For the `namespaces` resource itself, filtering applies unconditionally, regardless of used flags or headers. The `d8 k get namespaces` command returns only the namespaces available to the user instead of `403 Forbidden`. Filtering of arbitrary resources via `--scope` is a generalization of the same mechanism, enabled explicitly.

If the `console` module older than 1.44.0 is enabled, the namespace list is not filtered: the access levels of ClusterAuthorizationRule and AuthorizationRule and the namespace roles keep the permission to list every namespace.

The same list is returned by the read-only accessiblenamespaces resource, which any authenticated user can request for themselves:

```
d8 k get accessiblenamespaces
```

This is handy for scripts and interfaces: there is no need to iterate over namespaces and check access to each one.

Special case: projects

A `Project` is cluster-scoped, but its visibility follows the same namespace boundary: a user sees a project when they can see any of its namespaces — the main one or an additional one. A namespace belongs to a project only when its `projects.deckhouse.io/project` label has a non-empty value; the project is never derived from the namespace name. A namespace without the label shows the virtual project that `multitenancy-manager` counts it in: `deckhouse` for `d8-*`, `kube-*`, `upmeter-*` and namespaces with the `heritage: deckhouse` or `heritage: upmeter` label, `default` for the rest. A user who may list namespaces in the whole cluster sees every project, both virtual ones included. Filtering applies unconditionally here as well, so `d8 k get projects` returns the accessible subset instead of `403 Forbidden`, and a `get` of a project unavailable for the user answers `NotFound`.

Because of that, access levels granted through a `ClusterAuthorizationRule` no longer carry a cluster-wide read of `projects`: the grant would let RBAC allow the listing before the filter could narrow it. As a result, `d8 k auth can-i list projects` answers `no` for such a user while the listing itself works and returns their projects. Writing projects is unaffected.

Monitoring

kube-apiserver exports the following metrics of the mechanism:

Metric	Meaning

Metric	Meaning
`apiserver_scopefilter_allowed_set_failures_total`	Requests answered with an error because the access boundary could not be computed, by the `operation` label (`list`, `get`, `watch`) and the `reason` label: `unavailable` (`permission-browser` cannot answer now, the request got `503 Service Unavailable`), `no_resolvers` (kube-apiserver started without the mechanism wired for aggregated APIs, the request got `403 Forbidden`) or `denied` (any other failure, the request got `403 Forbidden`)
`apiserver_scopefilter_cluster_disagreements_total`	Requests kube-apiserver denied while `permission-browser` allowed them in the whole cluster, answered with `403 Forbidden`
`apiserver_scopefilter_watch_refresh_failures_total`	Failed refreshes of the namespace set of an open watch, timeouts included
`apiserver_scopefilter_watch_closed_total`	Watches closed by the mechanism, by the `reason` label: `refresh_failures` (the set could not be refreshed for 60 seconds), `overflow` (the client did not read the events in time), `multitenancy_disabled` (the `projects` and `project` scopes are no longer served) or `access_changed` (the user may now perform the request in the whole cluster)
`apiserver_scopefilter_proxied_namespaces`	Size of the namespace set sent to an aggregated API server with a scoped read
`apiserver_scopefilter_proxied_watch_closed_total`	Scoped watches of an aggregated API ended so that the client reconnects with a new set, by the `reason` label: `set_changed` (the set changed or the read is no longer filtered), `opt_in_withdrawn` (the `APIService` was deleted or no longer lists the resource) or `refresh_failed` (the set could not be recomputed for 60 seconds)
`apiserver scopefilter_proxied_header_limit_rejections_total`	Scoped reads of an aggregated API answered with `400 Bad Request` because the set does not fit the header limit

Components involved

Component	Role
Deckhouse CLI	Supports the `--scope` flag and transforms its values into headers
kube-apiserver (DP patch)	Bypasses `403 Forbidden` and provides filtering at the storage layer
`user-authz` (`permission-browser`)	Defines the access boundary via `SelfScopedAccessReview`
`multitenancy-manager`	Provides namespace classification based on the `projects.deckhouse.io/project` label
`user-authz` (the `d8-user-authz-multitenancy-state` ConfigMap)	Tells kube-apiserver whether the `multitenancy-manager` module is enabled, in every edition

Additional resources

- Module `multitenancy-manager` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with IaaS

Deckhouse Platform (DP) provides built-in tools for integrating with various cloud providers and virtualization systems. These tools let you:

- Automatically use cloud infrastructure to provision virtual machines and connect them to the cluster.
- Deploy clusters in cloud environments.
- Scale resources as needed.
- Automatically create base infrastructure resources (such as networks, routes, etc.), if supported by the provider.

The workflow for working with different cloud providers is generally the same, with only the preparation steps (such as creating a service account in the cloud) and configuration files differing.

Supported cloud providers:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- OpenStack
- OVH Cloud
- Selectel Cloud
- VK Cloud
- Yandex Cloud

Integration is also possible with private clouds based on the following solutions:

- VK Cloud
- OpenStack
- Huawei Cloud

In addition to cloud providers, integration is supported with the following virtualization systems:

- Deckhouse Virtualization Platform (DVP)
- VMware Cloud Director
- VMware vSphere
- zVirt

It is also possible to configure hybrid clusters.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Public cloud providers

Deckhouse Platform (DP) can be deployed in various public clouds, enabling the use of scalable infrastructure and flexible integration with cloud services.

DP supports the following providers:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- OpenStack
- OVH Cloud
- Selectel Cloud
- VK Cloud
- Yandex Cloud

This simplifies the choice of deployment environment and enables a unified approach to managing Kubernetes clusters across different cloud ecosystems.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Yandex Cloud

To allow Deckhouse Platform (DP) to manage resources in Yandex Cloud, you need to:

- Create a service account.
- Assign the required IAM roles to it.
- Generate an authorization key.
- Optionally reserve a public IP address.
- Ensure sufficient resource quotas in the cloud.

The steps below guide you through setting up the connection.

Preparing the environment

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

The `cloud-init` package must be installed on virtual machines (VM). After a VM is started, the following services associated with this package must be running:

- `cloud-config.service`
- `cloud-final.service`
- `cloud-init.service`

To verify that the services are running, use the following commands:

```
systemctl status cloud-config.service
systemctl status cloud-final.service
systemctl status cloud-init.service
```

> Starting with version 1.77, new clusters in Yandex Cloud use the Cilium CNI by default with pod traffic tunneling over VXLAN. In existing clusters, the CNI stays unchanged.
>
> Make sure all nodes run Linux kernel 5.8 or newer and that security groups allow UDP traffic between nodes. For the list of ports, see “Network interaction of the platform components”; for CNI settings, see “Internal network configuration”.

Creating a service account

To enable DP to manage Yandex Cloud resources, create a service account and assign it editing permissions. You can find detailed instructions on service account creation in the Yandex Cloud documentation.

To create the service account, run the following command:

```
yc iam service-account create --name deckhouse
```

The command will return information about the newly created service account:

```
id: <USER_ID>
folder_id: <FOLDER_ID>
created_at: "YYYY-MM-DDTHH:MM:SSZ"
name: deckhouse
```

⚠ Save the `userID` and `folderID` as they will be needed in the following steps.

Assigning IAM roles

To allow DP to work with cloud resources, assign the following roles to the service account:

```
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role compute.editor --subject
serviceAccount:<USER_ID>
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role vpc.admin --subject
serviceAccount:<USER_ID>
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role load-balancer.editor --subject
serviceAccount:<USER_ID>
```

Generating an authorization key

Generate a JSON authorization file to use in your configuration:

```
yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
```

The contents of the `deckhouse-sa-key.json` file are needed for the credentials Secret.

Credentials Secret

The service account key is stored in a separate Secret rather than in ModuleConfig. The provider reads it when starting the components that access the Yandex Cloud API.

The Secret is created during cluster installation together with the other resources of the initial configuration. If the provider is connected to a running cluster, the Secret is applied after the module creates the namespace. For the order of steps, see “Hybrid cluster with Yandex Cloud”. The Secret must meet the following requirements:

- The name is `d8-credentials` and the namespace is `d8-cloud-provider-yandex`.
- The type is `cloud-provider.deckhouse.io/credentials`.
- The `authScheme` field is set to `serviceAccount`.
- The `secret` field contains the service account key in JSON.

```
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-yandex
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: serviceAccount
  secret: |
    <SERVICE_ACCOUNT_JSON>
```

Replace `<SERVICE_ACCOUNT_JSON>` with the contents of the `deckhouse-sa-key.json` file.

To change the credentials, update the `secret` field:

```
d8 k -n d8-cloud-provider-yandex edit secret d8-credentials
```

In the “WithNATInstance layout”, the API key for monitoring the NAT instance is stored in a second Secret, `d8-credentials-exporter`, in the same namespace. Its `authScheme` field is set to `apiToken` and the key is placed in the `secret` field.

Instead of a ready key, the `secret` field can hold the `Auto` value. In that case, while provisioning the infrastructure, DP creates a service account named `<CLUSTER_PREFIX>-monitoring` in the folder, grants it the `monitoring.viewer` role, and generates an API key for it, which the metrics exporter then uses.

In clusters migrated to ModuleConfig, the service account key moves to the `d8-credentials` Secret from the `provider.serviceAccountJSON` parameter of the YandexClusterConfiguration resource. The

monitoring API key moves to the `d8-credentials-exporter` Secret from the `withNATInstance.exporterAPIKey` parameter.

Checking and increasing quotas

Make sure your cloud account has the required quotas for cluster deployment and scaling:

- vCPUs: 64
- SSD storage: 2000 GB
- Number of VMs: 25
- RAM: 256 GB

Increase quotas via the Yandex Cloud Console if necessary.

Reserving a public IP

If you are using the WithoutNAT or WithNATInstance deployment layout and need a fixed external IP address (for example, to use in `externalIPAddresses`, `natInstanceExternalAddress`, or a bastion host), run the following command:

```
yc vpc address create --external-ipv4 zone=ru-central1-a
```

Example output:

```
id: e9b4cfmmnc1mhgij75n7
folder_id: b1gog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
  requirements: {}
reserved: true
```

After completing these steps, you will have all the information needed to configure the provider. For the order of steps, see “Configuration”.

Additional resources

- Global custom resources
- Module `cloud-provider-yandex` provider configuration
- Module `cloud-provider-yandex` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in Yandex Cloud

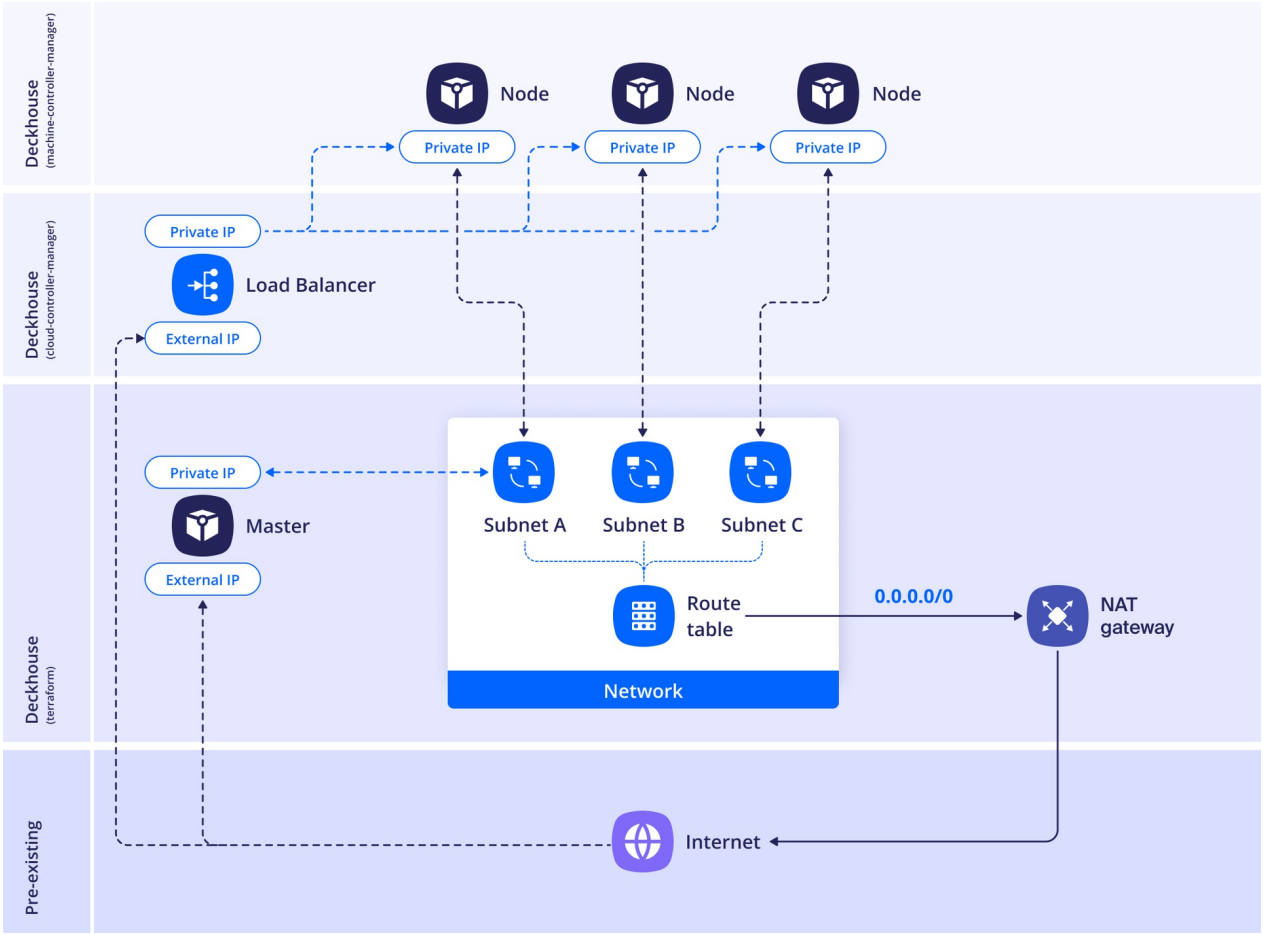
Layouts

This section describes the possible node placement layouts in Yandex Cloud infrastructure and the related configuration options. The layout is specified by the `layout` parameter in the ModuleConfig resource of the `cloud-provider-yandex` module. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

✖ In this layout, nodes will not have public IP addresses and will access the internet via a Yandex Cloud NAT Gateway. The NAT Gateway uses random public IP addresses from dedicated address ranges. Because of this, you cannot whitelist the IP addresses of cloud resources behind a specific NAT Gateway in third-party services.

⚠ Because nodes are created without public IP addresses in this layout, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly over a private network or through a bastion host. If the master node is not directly accessible, run the installation with the `--ssh-bastion-host`, `--ssh-bastion-user`, and, if needed, `--ssh-bastion-port` parameters.



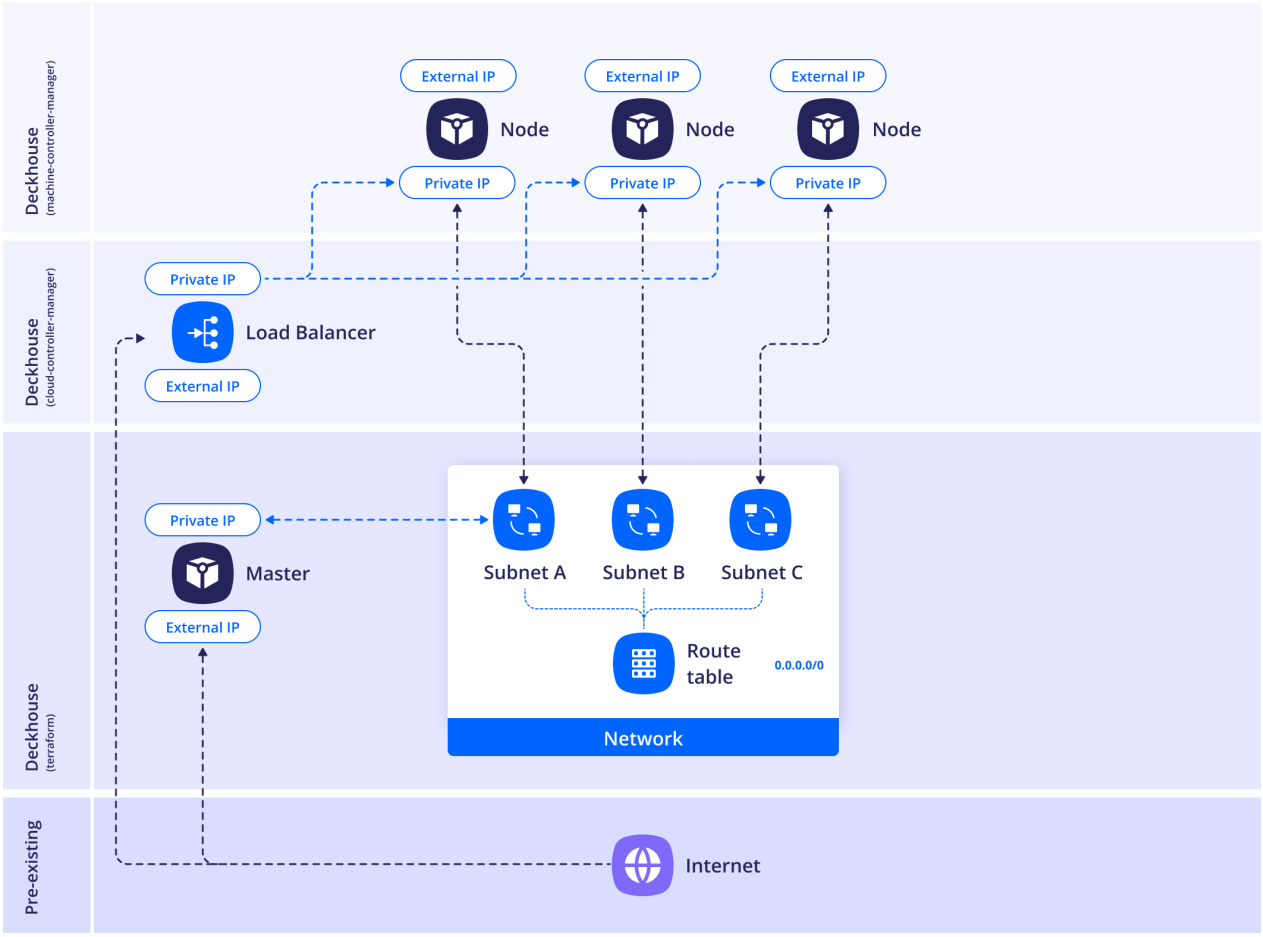
Example layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
          - ru-central1-b
          - ru-central1-d
        labels:
          billing: prod
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

WithoutNAT

In this layout, no form of NAT is used, and each node is assigned a public IP address.

⚠ DP does not assign security groups to the nodes, so in this layout the nodes are accessible from the outside without restrictions. To restrict the traffic, use the rules of the default group as described in “Configuring security groups in Yandex Cloud”.



Example layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: WithoutNAT
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
          - ru-central1-b
          - ru-central1-d
        externalIPAddresses:
          master:
            - Auto
            - Auto
            - Auto
          worker:
            - <ZONE_A_EXTERNAL_IP_WORKER_1>
            - Auto
        externalSubnetIDs:
          master:
            - <ZONE_A_SUBNET_ID>
            - <ZONE_B_SUBNET_ID>
            - <ZONE_D_SUBNET_ID>
          worker:
            - <ZONE_A_SUBNET_ID>
            - <ZONE_B_SUBNET_ID>
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

WithNATInstance

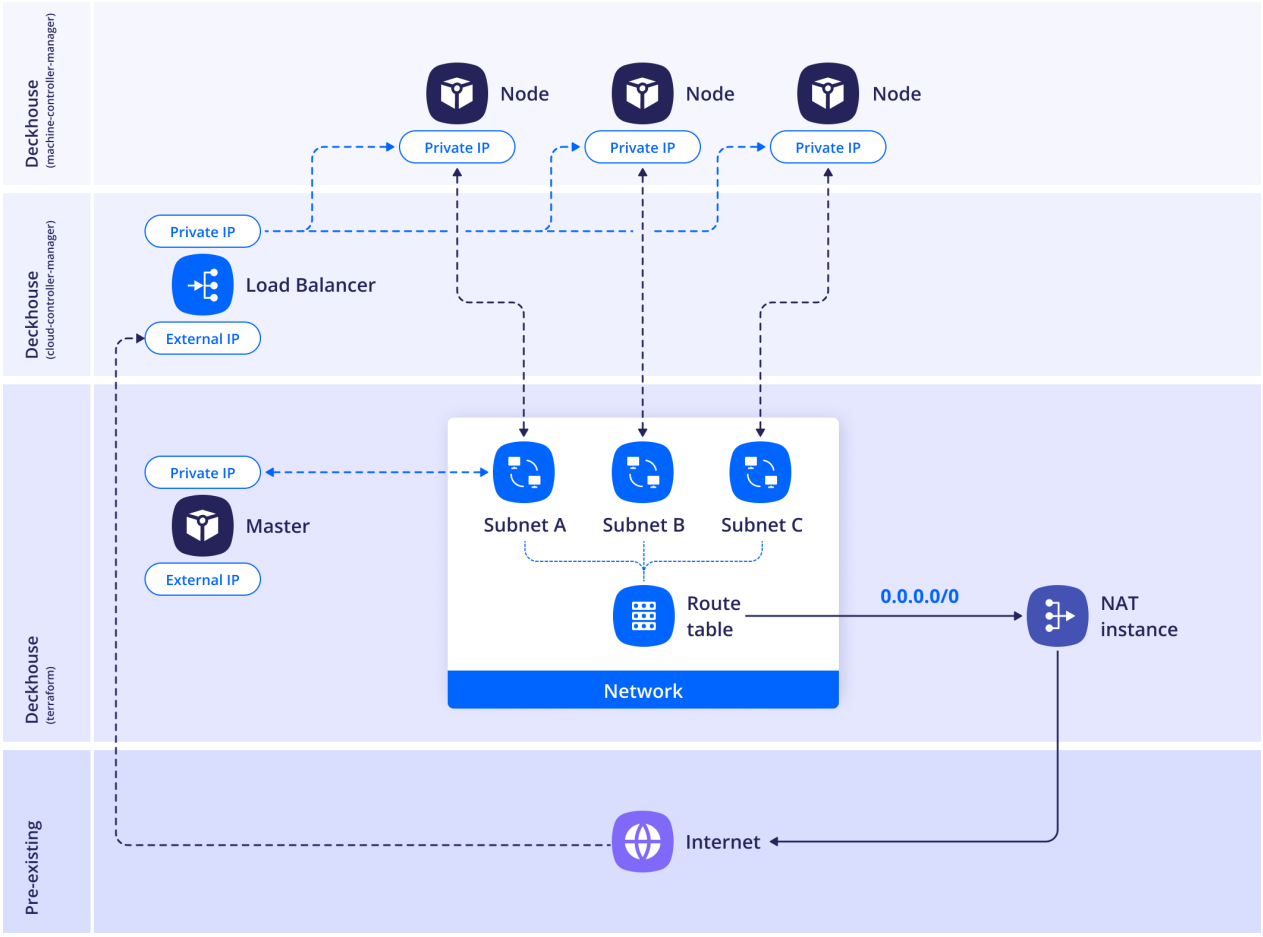
In this layout, a NAT instance is created in a separate subnet, and the route tables of the zone subnets are updated with a `0.0.0.0/0` route using the NAT instance as the next hop. The subnet is isolated to prevent routing loops and must not overlap with any other networks used in the cluster.

To place the NAT instance in an existing subnet, use the `withNATInstance.internalSubnetID` parameter. The instance will be created in the zone corresponding to this subnet.

If you need to create a new subnet, specify the `withNATInstance.internalSubnetCIDR` parameter. The NAT instance will be placed in this subnet.

Set one of the two parameters, `withNATInstance.internalSubnetID` or `withNATInstance.internalSubnetCIDR`.

If `withNATInstance.externalSubnetID` is also specified, the NAT instance will be connected to that subnet via a secondary network interface.



Example layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: WithNATInstance
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
        withNATInstance:
          natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
          internalSubnetID: <INTERNAL_SUBNET_ID>
          externalSubnetID: <EXTERNAL_SUBNET_ID>
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

> **i** In this example, master and worker nodes are created without public IP addresses and without additional external network interfaces. Outbound traffic from the cluster subnets goes through the NAT instance.
>
> The NAT instance is not automatically used by the `dhctl` installer as a bastion host or jump host. To install the cluster, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly or through a bastion host.
>
> If the master node does not have a public IP address or is not directly accessible, run the installation with the `--ssh-bastion-host` and `--ssh-bastion-user` parameters.
>
> If you need to connect directly to a master node using a public IP address, specify the `externalIPAddresses` and `externalSubnetIDs` parameters in the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module. Note that the interface connected to the network specified in `externalSubnetIDs` will be used as the node's default gateway.

Configuration

The Yandex Cloud provider parameters are defined in four resources:

- The ModuleConfig of the `cloud-provider-yandex` module defines the layout, the network parameters, and the cloud and folder IDs.
- The `d8-credentials` Secret in the `d8-cloud-provider-yandex` namespace stores the service account key. Its format is described in "Credentials Secret".
- YandexInstanceClass describes the virtual machine parameters.
- NodeGroup describes the node groups and refers to an instance class. The master node group is described by the same resource with the `CloudPermanent` type.

To change the provider settings in a running cluster, edit the ModuleConfig:

```
d8 k edit mc cloud-provider-yandex
```

After changing the node parameters, run the following command:

```
dhctl converge
```

> **⚠** If the cluster was installed with the YandexClusterConfiguration schema, it must be migrated to the ModuleConfig-based configuration. Until the migration is complete, the `D8CloudProviderYandexCloudMigrationPending` alert fires and a Deckhouse Platform update can be blocked.
>
> For instructions, see "How to migrate a cloud provider to ModuleConfig-based configuration?".
>
> If the module settings had the `storageClass.default` parameter, set the default StorageClass with the global `defaultClusterStorageClass` parameter after the migration. The conversion does not carry this value over.

Configuration example

Below is a minimal configuration of a cluster with the Standard layout, one master node, and one group of worker nodes:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
```

```
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 10.241.32.0/20
        sshPublicKey: <SSH_PUBLIC_KEY>
        labels:
          billing: prod
---
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: master
spec:
  cores: 4
  memory: 8192
  imageID: <IMAGE_ID>
  etcdDiskSizeGB: 10
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: YandexInstanceClass
      name: master
    minPerZone: 1
    maxPerZone: 1
---
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: worker
spec:
  cores: 4
  memory: 8192
  coreFraction: 50
  imageID: <IMAGE_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: YandexInstanceClass
      name: worker
    minPerZone: 1
    maxPerZone: 3
    zones:
      - ru-central1-a
```

Where:

- `CLOUD_ID` and `FOLDER_ID` are the Yandex Cloud cloud and folder identifiers, specified by the `cloudID` and `folderID` parameters in the `provider.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module. For instructions on obtaining these identifiers, see “Connecting and authorizing in Yandex Cloud”.
- `SSH_PUBLIC_KEY` is the public SSH key used to access nodes, specified by the `sshPublicKey`

parameter in the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module.
- `IMAGE ID` is the operating system image ID used to create virtual machines, specified by the `imageID` parameter of the YandexInstanceClass resource. If the parameter is not set, the image of the `master` node group instance class is used.

The YandexInstanceClass resource for the `master` node group must specify the `etcdDiskSizeGB` parameter, which defines the size of a separate disk for etcd. This parameter must not be specified for other node groups.

The rest of the parameters are described in the module settings and in the YandexInstanceClass resource description.

The service account key is not stored in ModuleConfig. It is defined in a separate Secret.

Network parameters and security

This section describes the settings related to addressing, routing, external traffic, and network security in a DP cluster deployed in Yandex Cloud.

Internal node addressing

The `nodeNetworkCIDR` parameter defines the IP address range that is split between availability zones and applied to the internal node interfaces. The parameter belongs to the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  settings:
    nodes:
      parameters:
        nodeNetworkCIDR: 192.168.12.0/24
```

Deckhouse Platform divides `nodeNetworkCIDR` into four equal parts and creates internal subnets for nodes in the corresponding availability zones. Each default zone has a fixed part of the range:

Zone	Part of the range
`ru-central1-a`	First
`ru-central1-b`	Second
`ru-central1-e`	Third
`ru-central1-d`	Fourth

You can list up to four zones in `zones`. If the parameter is not set, Deckhouse Platform uses all four zones in the table.

Instructions for using the `ru-central1-f` zone are provided in “Using the ru-central1-f zone”.

Nodes connect to the subnet in their zone. The range is divided this way in every layout. If `existingZoneToSubnetIDMap` is set, Deckhouse Platform uses the mapped subnets without creating new ones.

> **i** A subnet accepts only one route table, so two clusters with the `cni-simple-bridge` module cannot use the same subnets. To deploy such clusters in shared subnets, use the `cni-cilium` module.

Assigning external IP addresses and outgoing traffic

The `externalSubnetIDs` parameter belongs to the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module and is set per node group:

- the key is the node group name;

- the value is the list of Yandex Cloud subnet IDs that the external network interfaces of that group's nodes are attached to, in the order of the zones.

The parameter is required when you need to:

- assign a public IP address;
- define a default route for outgoing traffic from nodes;
- use the `Auto` value in the `externalIPAddresses` parameter.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  settings:
    nodes:
      parameters:
        externalSubnetIDs:
          worker:
            - <RU_CENTRAL1_A_SUBNET_ID>
            - <RU_CENTRAL1_B_SUBNET_ID>
            - <RU_CENTRAL1_D_SUBNET_ID>
```

The `externalIPAddresses` parameter in the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module specifies a list of external IP addresses for each node group in the order of their availability zones.

DNS and DHCP settings for internal networks

The `dhcpOptions` parameter sets the DHCP server configuration applied to all subnets created within the DP cluster in Yandex Cloud. The parameter belongs to the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module.

Available fields:

- `domainName`: The search domain set in the network configuration.
- `domainNameServers`: An array of IP addresses of DNS servers used as recursive resolvers.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  settings:
    nodes:
      parameters:
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - 192.168.0.2
            - 192.168.0.3
```

i The DNS servers listed in `domainNameServers` must resolve both external and internal domain zones used by the cluster. Otherwise, service disruptions may occur.

After updating `dhcpOptions`:

- force renew the DHCP lease on all VMs;
- restart the components running in the node network (`hostNetwork: true`), including `kube-dns`,

to refresh the `resolv.conf` contents.

You can apply changes using the following command:

```
netplan apply
```

or another method used in your system, for example `systemd-networkd` or `dhclient`.

Using pre-existing subnets

The `existingZoneToSubnetIDMap` parameter maps availability zones to pre-created subnets in Yandex Cloud. Set it when DP has to use existing subnets instead of creating new ones. The parameter belongs to the `nodes.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  settings:
    nodes:
      parameters:
        existingZoneToSubnetIDMap:
          ru-central1-a: e2lu8r1tbbtryhdpa9ro
          ru-central1-b: e2lu8r1tbbtryhdpa9ro
          ru-central1-d: e2lu8r1tbbtryhdpa9ro
```

i DP creates a route table but does not associate it with the subnets. Associate the table manually in the Yandex Cloud interface.

Additional external networks

The `additionalExternalNetworkIDs` parameter defines a list of additional external networks whose addresses DP treats as public. The parameter belongs to the `ccm.parameters` section of the ModuleConfig resource of the `cloud-provider-yandex` module.

This is useful when:

- There are external subnets not listed in `externalSubnetIDs` but that should be treated as external.
- You need precise control over how IP addresses are interpreted for load balancing, routing, and status export.
- You use custom network configurations (for example, external NAT gateways or manually reserved IPs).

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    ccm:
      parameters:
        additionalExternalNetworkIDs:
          - enp6t4sno
```

If this parameter is not set, DP will only use the subnets explicitly specified in the module settings (for example, via `externalSubnetIDs`) to determine whether an IP is public.

Configuring security groups in Yandex Cloud

For every cloud network, Yandex Cloud creates a default security group, including for the cluster network. That group allows any inbound and outbound traffic and applies to the virtual machine network interfaces that are not assigned another group.

You can restrict the cluster traffic with the rules of that group. Assigning another security group to the nodes is not possible, as neither the module settings nor the YandexInstanceClass resource have parameters for that.

⚠ Do not assign security groups to the nodes manually in the Yandex Cloud console. The `dhctl converge` command brings the virtual machines to the state described in the cluster configuration and may remove such an assignment.

✖ Do not delete the rules that allow any traffic until you have configured the other rules of the group. Otherwise, the group blocks the traffic between the nodes and the cluster components lose connection to each other.

Errors in the rules disrupt the cluster. Before configuring the rules in a production cluster, consult the security group usage details in Yandex Cloud.

If the cluster uses Cilium with pod traffic tunneling over VXLAN, allow UDP traffic between nodes on the ports listed in “Network interaction of the platform components”.

To configure the rules of the security group:

1. Identify the cloud network used by the DP cluster.

   The network name matches the `prefix` parameter of the ModuleConfig `global`. You can retrieve it using the following command:

   ```
   d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
   ```

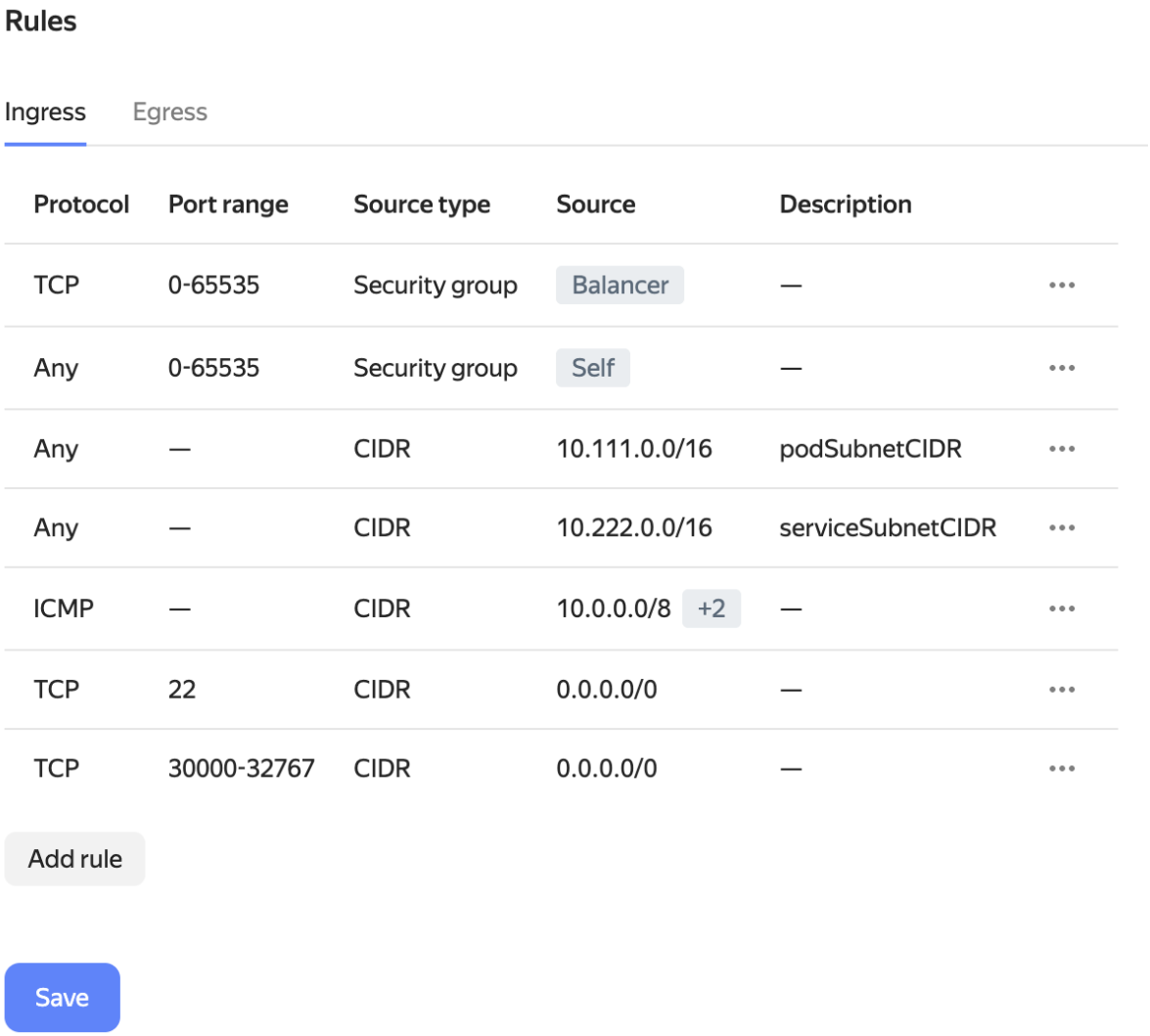
   If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

   ```
   d8 k get secrets -n kube-system d8-cluster-configuration -ojson | \
     jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
   ```

2. In the Yandex Cloud Console, go to the Virtual Private Cloud service and open the “Security Groups” section. You should see a security group marked `Default`.

3. Create rules according to the Yandex Cloud instructions.



4. Delete the rule that allows all **inbound** traffic (it is already deleted in the screenshot above) and save your changes.

Configuring access via bastion host

To connect to nodes located in private subnets (for example, when using the Standard or WithNATInstance layout), you can use a bastion host, an intermediary VM with a public IP address that provides SSH access to the nodes.

To configure access, follow these steps:

1. Bootstrap the base infrastructure. Before creating a bastion host, perform the initial installation phase of DP, which sets up the network infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config.yml
   ```

2. Create a bastion host in Yandex Cloud:

   ```
   yc compute instance create \
     --name bastion \
     --hostname bastion \
     --create-boot-disk image-family=ubuntu-2204-lts,image-folder-id=standard-
   images,size=20,type=network-hdd \
     --memory 2 \
     --cores 2 \
     --core-fraction 100 \
     --ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
     --zone ru-central1-a \
     --public-address 178.154.226.159
   ```

   Replace `<SSH_PUBLIC_KEY_FILE>` with the name of the public key, for example `id_rsa.pub` for

an RSA key or `id_ed25519.pub` for an ED25519 key.

Ensure the IP address specified in `--public-address` is accessible from your network and is correct.

3. Start the main DP bootstrap through the bastion host:

```
dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
  --ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --
config=/config.yml
```

Replace `<SSH PRIVATE KEY_FILE>` with the name of the private key, for example `id_rsa` for an RSA key or `id_ed25519` for an ED25519 key.

Where:

- `--ssh-bastion-user`: User for connecting to the bastion host.
- `--ssh-user`: User on the target cluster nodes.
- `--ssh-agent-private-keys`: Path to the private SSH key.
- `--config`: Path to the DP configuration file.

Replacing the OS image

The `imageID` parameter of YandexInstanceClass defines the image for `CloudEphemeral` nodes. With a ModuleConfig-based provider configuration, this parameter also defines the image for `CloudPermanent` nodes and master nodes.

With a YandexClusterConfiguration-based configuration, the image for `CloudPermanent` nodes and master nodes is set by `imageID` in `nodeGroups[].instanceClass` and `masterNodeGroup.instanceClass`, respectively.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

Additional resources

- Global custom resources
- Global parameters
- Module `cloud-provider-yandex` documentation
- Module `cni-cilium` documentation
- Module `cni-simple-bridge` documentation
- Module `node-manager` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in Yandex Cloud

This section covers additional aspects of Deckhouse Platform (DP) integration with Yandex Cloud:

- Attaching cloud disks via CSI.
- Automatic StorageClass creation.
- Use of load balancers.
- Specifics of applying changes.
- Working with CloudStatic nodes and bastion hosts.

Storage (CSI and StorageClass)

DP integrates with Yandex Cloud block storage via the Container Storage Interface (CSI) component. This allows DP clusters to automatically provision and attach disks and use standard Kubernetes PersistentVolumeClaim resources for working with the storage.

DP automatically creates StorageClass resources for all supported Yandex Cloud disk types, enabling immediate storage usage without the need to define classes manually.

The following disk types are supported:

Disk type	StorageClass name	Notes
`network-hdd`	`network-hdd`	—
`network-ssd`	`network-ssd`	—
`network-ssd-nonreplicated`	`network-ssd-nonreplicated`	Size must be a multiple of 93 GB
`network-ssd-io-m3`	`network-ssd-io-m3`	Size must be a multiple of 93 GB

i The sizes of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be multiples of 93 GB. Otherwise, volume provisioning will fail.

Excluding unnecessary StorageClasses

If certain disk types won't be used in the cluster, you can disable automatic creation of the corresponding StorageClass objects using the `settings.storage.parameters.excludedStorageClasses` parameter in the ModuleConfig resource:

```
settings:
  storage:
    parameters:
      excludedStorageClasses:
        - network-ssd-.*
        - network-hdd
```

In this example, DP will not create StorageClass resources for any `network-ssd` or `network-hdd` disks.

Creating additional StorageClasses and block size

The `settings.storage.parameters.provisionedStorageClasses` parameter lets you create additional StorageClasses or override the parameters of StorageClasses created by DP by default.

Use the `blockSize` parameter to set the block size for provisioned disks. The block size determines the maximum disk size. It is `8Ti` for `4Ki` and doubles with each next block size, up to `256Ti` for `128Ki`.

Example configuration of the `cloud-provider-yandex` module with an additional StorageClass configured with a block size of `64Ki`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 192.168.12.0/24
        sshPublicKey: <SSH_PUBLIC_KEY>
    storage:
      parameters:
        provisionedStorageClasses:
          - name: network-ssd-64k
            type: network-ssd
            blockSize: 64Ki
```

⚠ After a disk is created, its block size cannot be changed. Changing the `blockSize` parameter recreates the StorageClass, but does not change the block size of previously provisioned volumes.

Setting the default StorageClass

By default, DP determines the StorageClass using the `storageclass.kubernetes.io/is-default-class=true` annotation.

To explicitly set a different default StorageClass, use the global DP parameter `global.defaultClusterStorageClass`. You can modify it with the following command:

```
kubectl edit mc global
```

If `defaultClusterStorageClass` is not specified, DP determines the default StorageClass in the following order:

- A StorageClass with the annotation `storageclass.kubernetes.io/is-default-class='true'` (if it exists in the cluster).
- The first StorageClass in alphabetical order auto-created by the cloud provider.
- The default value of `defaultClusterStorageClass` is an empty string (`""`).

Resizing a PVC

You can increase the size of an existing PVC by changing the value of the `spec.resources.requests.storage` parameter, without stopping or recreating the pod that uses it.

After the value of `spec.resources.requests.storage` is changed, the CSI driver performs the following operations:

- increases the size of the disk in Yandex Cloud;
- updates the size of the associated PersistentVolume;
- expands the file system on the node to which the volume is attached.

During the operation, the pod continues running, and the mounted volume remains available to the application. After the resize operation is complete, the new file system size becomes available inside the container without restarting the pod.

ℹ Decreasing the size of a PVC is not supported.

To increase the size of a PVC, follow these steps:

1. Get the name of the StorageClass used by the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME> \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the PVC;
   - `<PVC_NAME>` — the name of the PVC to resize.

   For example:

   ```
   d8 k -n production get pvc application-data \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Example output:

   ```
   network-ssd
   ```

   In this example, the `application-data` PVC uses the `network-ssd` StorageClass.

2. Make sure that the StorageClass allows volume expansion:

```
d8 k get storageclass <STORAGECLASS_NAME> \
  -o jsonpath='{.allowVolumeExpansion}{"\n"}'
```

Where `<STORAGECLASS_NAME>` is the name of the StorageClass obtained in the previous step.

For example:

```
d8 k get storageclass network-ssd \
  -o jsonpath='{.allowVolumeExpansion}{"\n"}'
```

Example output:

```
true
```

3. Check the current status and size of the PVC:

```
d8 k -n <NAMESPACE> get pvc <PVC_NAME>
```

For example:

```
d8 k -n production get pvc application-data
```

Example output:

```
NAME               STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi
RWO            network-ssd
```

Make sure that:

- the PVC is in the `Bound` state;
- the `CAPACITY` field shows the current PVC size;
- the `STORAGECLASS` field shows the StorageClass checked in the previous step.

4. Increase the PVC size:

```
d8 k -n <NAMESPACE> edit pvc <PVC_NAME>
```

For example:

```
d8 k -n production edit pvc application-data
```

Specify the new PVC size in the `spec.resources.requests.storage` field:

```
spec:
  resources:
    requests:
      storage: 30Gi
```

In this example, the PVC size is increased to `30Gi`.

Save the changes and close the editor.

> ⚠ For the `network-ssd-nonreplicated` and `network-ssd-io-m3` StorageClasses, the size must be a multiple of 93Gi.

5. Wait for the PVC resize operation to complete:

```
d8 k -n <NAMESPACE> get pvc <PVC_NAME> --watch
```

Where:

- `<NAMESPACE>` — the namespace containing the PVC;
- `<PVC_NAME>` — the name of the PVC being resized.

For example:

```
d8 k -n production get pvc application-data --watch
```

During the resize operation, the `CAPACITY` field may continue to show the previous size:

```
NAME               STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi
RWO            network-ssd
```

The operation is complete when the `CAPACITY` field shows the new PVC size:

```
NAME               STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   30Gi
RWO            network-ssd
```

6. Check the PVC events:

```
d8 k -n <NAMESPACE> describe pvc <PVC_NAME>
```

For example:

```
d8 k -n production describe pvc application-data
```

The following events may appear during the resize operation:

```
ExternalExpanding
Resizing
FileSystemResizeRequired
```

The following event indicates that the file system was successfully expanded:

```
FileSystemResizeSuccessful
```

For example:

```
Normal  FileSystemResizeSuccessful  kubelet  MountVolume.NodeExpandVolume
succeeded for volume "pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
```

7. Check the file system size inside the pod:

```
d8 k -n <NAMESPACE> exec <POD_NAME> -- \
  df -hT <MOUNT_PATH>
```

Where:

- `<NAMESPACE>` — the namespace containing the pod;
- `<POD_NAME>` — the name of the pod using the PVC;
- `<MOUNT_PATH>` — the path inside the container where the PVC is mounted.

For example:

```
d8 k -n production exec application-0 -- \
  df -hT /data
```

Example output:

```
Filesystem   Type   Size    Used   Avail   Use%   Mounted on
/dev/vde     ext4   29.4G   22M    29.4G    1%    /data
```

> **i** The actual file system size may be slightly smaller than the PVC size due to file system metadata.

Load balancing

External LoadBalancer

DP automatically watches for Kubernetes Service objects of type LoadBalancer. When such a service is created, DP creates the following resources:

- **NetworkLoadBalancer**: A Yandex Cloud network load balancer.
- **TargetGroup**: A group of endpoints for traffic distribution.

These resources allow Kubernetes LoadBalancer services to receive traffic from the internet or internal networks, depending on the configuration.

For more details on the architecture, refer to the Kubernetes Cloud Controller Manager for Yandex Cloud documentation.

Internal LoadBalancer

To create an internal load balancer, you must explicitly specify the subnet where the load balancer listener should be created.

To do this, add the following annotation to the Service object:

```
metadata:
  annotations:
    yandex.cpi.flant.com/listener-subnet-id: <SUBNET_ID>
```

`SubnetID` refers to the ID of the subnet where the internal listener for the Yandex LoadBalancer will be created. This allows you to control the load balancer’s network exposure and limit it to internal addresses only.

> **i** The default behavior (external or internal LB) depends on the cluster configuration. To explicitly choose the type, use the `yandex.cpi.flant.com/loadbalancer-external` annotation.

Service annotations

Default values are configured in the cluster for placing load balancer resources (the network for the target group and the subnet for the Listener). These values are set automatically during cluster setup and can be overridden with annotations at the individual Service level.

The following annotations are supported by Yandex Cloud Controller Manager:

1. `yandex.cpi.flant.com/target-group-network-id` — specifies the NetworkID in which the target group for this Service will be created. Overrides the corresponding default value.
2. `yandex.cpi.flant.com/listener-subnet-id` — sets the SubnetID for the Listeners of the LB created for this Service. Overrides the corresponding default value.
3. `yandex.cpi.flant.com/listener-address-ipv4` — sets a predefined IPv4 address for the Listeners (supported for both internal and external LBs).
4. `yandex.cpi.flant.com/loadbalancer-external` — enables creation of an external LB for this Service (use it when you need to explicitly create an external load balancer). Overrides the default behavior.
5. `yandex.cpi.flant.com/target-group-name-prefix` — specifies the name prefix of the target group used by the LoadBalancer. The annotation on the Service identifies the target group but does not select nodes. To include nodes in the target group, set the same annotation value in `spec.nodeTemplate.annotations` in the NodeGroup. Yandex Cloud Controller Manager includes all suitable nodes carrying this annotation value, regardless of their NodeGroup. The target group name is formed as `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`.

If separate target groups are created for the control plane or master nodes, add the label `node.kubernetes.io/exclude-from-external-load-balancers: ""` to the master nodes. This prevents the controller from automatically adding master nodes to new target groups for load balancers. If you create your own load balancer for master nodes and want YCC to also be able to place its load balancers on master nodes, pre-create a target group with a name matching the pattern `${CLUSTER-NAME}${VPC.ID}`.

Using a separate target group for a NodeGroup

By default, Yandex Cloud Controller Manager adds all suitable cluster nodes to the default target group. To assign nodes to a separate target group, use the `yandex.cpi.flant.com/target-group-name-prefix` annotation.

1. In the NodeGroup whose nodes should be included in a separate target group, specify the `yandex.cpi.flant.com/target-group-name-prefix` annotation in the `spec.nodeTemplate.annotations` parameter. For example:

   ```
   spec:
     nodeTemplate:
       annotations:
         yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation is propagated to the Node objects. Yandex Cloud Controller Manager adds all suitable nodes carrying the same annotation value to the target group, regardless of their NodeGroup.

2. In the Service object of the LoadBalancer type, specify the same annotation with the same value. For example:

   ```
   metadata:
     annotations:
       yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation on the Service identifies the target group used by the LoadBalancer but does not select nodes.

3. Make sure that the values of `yandex.cpi.flant.com/target-group-name-prefix` in the NodeGroup and Service match. In the example above, both resources use the `frontend-` value.

4. The target group will be created with a name generated according to the following pattern:

   ```
   <ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
   ```

   For example, if the annotation value is `frontend-`, the cluster name is `my-cluster-`, and the network ID is `enp123456789`, the target group name will be:

   ```
   frontend-my-cluster-enp123456789
   ```

Configuration example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer
```

⚠ Yandex Cloud does not allow a single target, identified by the (`SubnetID, IP`) pair, to belong to multiple target groups at the same time.

When the target group is changed, Yandex Cloud Controller Manager first removes the target from the current group and then adds it to the new one. This transition may cause a brief interruption in traffic handling.

Nodes with a dedicated target group prefix are excluded from the default target group. As a result, load balancers that use the default target group stop routing traffic to those nodes.

This automatic transition is performed only between target groups managed by Yandex Cloud Controller Manager. If a target already belongs to another target group that is not managed by Yandex Cloud Controller Manager, remove it from that group before using the annotation.

After the prefix is changed or removed, the previous target group may remain empty. If it is no longer used by any load balancers, delete it manually.

Target group health checks

Health check parameters (for LB target groups created by the controller):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — how often to run the check, in seconds (default: 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — how long to wait for an endpoint response, in seconds. If no response is received within this time, the check is considered failed (default: 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — how many consecutive failed checks are required to mark an endpoint as unhealthy and exclude it from load balancing (default: 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — how many consecutive successful checks are required to return an endpoint to healthy status and include it back in load balancing (default: 2).

Applying changes

DP does not recreate existing Machine objects when configuration parameters change. Node recreation occurs only when:

- NodeGroup section parameters change.
- YandexInstanceClass parameters change.

This behavior helps prevent unnecessary operations and node idling, but it means you must manually recreate VMs if needed.

If you change the provider settings in the `cloud-provider-yandex` ModuleConfig (for example, the provider parameters, the layout, or the subnets), run the following command to apply the changes:

```
dhctl converge
```

This command reconciles the cluster state with the configuration defined in the resources.

Changing `networkType` does not recreate CloudEphemeral nodes

In clusters that use Machine Controller Manager (MCM), changing the `networkType` parameter in a YandexInstanceClass resource does not trigger automatic recreation of existing CloudEphemeral nodes. Although the MachineClass is updated, the network acceleration type of existing virtual machines in Yandex Cloud remains unchanged.

DP intentionally does not take the `networkType` parameter into account when determining whether CloudEphemeral nodes should be recreated in MCM. If it did, updating DP would recreate CloudEphemeral nodes in all clusters where `networkType` is specified, even if its value had not changed.

In Cluster API, the `networkType` parameter is taken into account when determining whether nodes should be recreated, so changing it automatically triggers node recreation. New CloudEphemeral NodeGroups in Yandex Cloud use Cluster API by default.

If you changed `networkType` in an MCM-based cluster and want the new value to apply to existing CloudEphemeral nodes, recreate them manually — instructions in the documentation.

Integrating manually created VMs

DP allows you to connect existing VMs in Yandex Cloud to the Kubernetes cluster as nodes. These nodes are called CloudStatic and they are not directly managed by the `node-manager` module but can still be used in the cluster.

To manually connect a VM as a CloudStatic node, follow these steps:

1. Retrieve the current `nodeNetworkCIDR` value from the cluster:

   ```
   d8 k get mc cloud-provider-yandex -o
   jsonpath='{.spec.settings.nodes.parameters.nodeNetworkCIDR}{"\n"}'
   ```

   Expected output:

   ```
   192.168.12.0/24
   ```

   Copy this value and use it in the VM metadata as `value`.

2. Set the `node-network-cidr` parameter in the VM metadata:

   ```
   key: node-network-cidr
   value: <NODE_NETWORK_CIDR>
   ```

   The `node-network-cidr` parameter must match the value of the `nodeNetworkCIDR` parameter in the ModuleConfig.

Additional resources

- Global parameters
- Global custom resources
- Module `cloud-provider-yandex` documentation
- Module `node-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with Yandex Cloud services

Deckhouse Platform (DP) supports native integration with several Yandex Cloud services. This allows for external monitoring, secure secret management, and automated data synchronization between the infrastructure and the cluster.

This section describes steps to configure the following integrations:

- Yandex Lockbox via External Secrets Operator
- Yandex Managed Service for Prometheus

Integration with Yandex Lockbox

DP supports integration with Yandex Lockbox using the External Secrets Operator (ESO). This allows Kubernetes secrets in the cluster to be automatically synchronized with Lockbox secrets.

To set up the integration:

1. Create a service account for External Secrets Operator:

   ```
   yc iam service-account create --name eso-service-account
   ```

2. Create an authorized key for the service account and save it to a file:

   ```
   yc iam key create --service-account-name eso-service-account --output authorized-key.json
   ```

3. Assign `lockbox.editor`, `lockbox.payloadViewer`, and `kms.keys.encrypterDecrypter` roles to the service account for accessing all secrets in the folder:

   ```
   folder_id=<folderID>
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name
   eso-service-account --role lockbox.editor
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name
   eso-service-account --role lockbox.payloadViewer
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name
   eso-service-account --role kms.keys.encrypterDecrypter
   ```

   For fine-grained access control, refer to the Yandex Lockbox access documentation.

4. Install External Secrets Operator using the Helm chart as described in this guide.

   Note that you may need to specify additional parameters such as `nodeSelector`, `tolerations`, etc. To do this, use the `./external-secrets/values.yaml` file after extracting the Helm chart.

   Download and extract the chart using the following command:

   ```
   helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
     --version 0.5.5 \
     --untar
   ```

   Install the chart:

   ```
   helm install -n external-secrets --create-namespace \
     --set-file auth.json=authorized-key.json \
     external-secrets ./external-secrets/
   ```

   Where:

   - `authorized-key.json`: The file created in step 2.

5. Create a SecretStore resource containing the `sa-creds` secret:

```
kubectl -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: SecretStore
metadata:
  name: secret-store
spec:
  provider:
    yandexlockbox:
      auth:
        authorizedKeySecretRef:
          name: sa-creds
          key: key
EOF
```

Where:

- `sa-creds`: Name of the Secret containing the authorized key. The secret appears after the Helm chart is installed.
- `key`: Name of the key in the `.data` field of the secret.

6. Verify that External Secrets Operator is working:

   ```
   kubectl -n external-secrets get po
   ```

   Example output:

   ```
   NAME                                                READY   STATUS    RESTARTS
   AGE
   external-secrets-55f78c44cf-dbf6q                   1/1     Running   0
   77m
   external-secrets-cert-controller-78cbc7d9c8-rszhx   1/1     Running   0
   77m
   external-secrets-webhook-6d7b66758-s7v9c            1/1     Running   0
   77m
   ```

   Check the SecretStore status as well:

   ```
   kubectl -n external-secrets get secretstores.external-secrets.io
   ```

   Example output:

   ```
   NAME           AGE   STATUS
   secret-store   69m   Valid
   ```

7. Create a secret in Yandex Lockbox with the following parameters:

   - Name: `lockbox-secret`.
   - Key: Enter non-confidential ID `password`.
   - Value: Enter confidential data for storing `p@$$w0rd`.

8. Create an ExternalSecret pointing to the `lockbox-secret` in the `secret-store`:

```
kubectl -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
  - secretKey: password
    remoteRef:
      key: <SECRET_ID>
      property: password
EOF
```

Where:

- `target.name`: Name of the new secret. External Secret Operator will create this secret in the DP cluster and put the `lockbox-secret` Yandex Lockbox secret parameters into it.
- `data[].secretKey`: Name of the key in the `.data` field of the secret created by External Secret Operator.
- `data[].remoteRef.key`: ID of the `lockbox-secret` Yandex Lockbox secret created earlier (for example, `e6q28nvfmhu539******`).
- `data[].remoteRef.property`: Key for the `lockbox-secret` Yandex Lockbox secret.

9. Verify that the new `k8s-secret` key contains the `lockbox-secret` secret value:

```
kubectl -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

The output will contain the `password` key value of the `lockbox-secret` secret created earlier:

```
p@$$w0rd
```

Integration with Yandex Managed Service for Prometheus

This integration lets you use Yandex Managed Service for Prometheus as an external storage for metrics (for example, for long-term retention).

To configure PrometheusRemoteWrite, follow these steps:

1. Create a service account with the `monitoring.editor` role.
2. Create an API key for the service account.
3. Create a PrometheusRemoteWrite resource:

```
kubectl apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <REMOTE_WRITE_URL>
  bearerToken: <API_KEY>
EOF
```

Where:

- `<REMOTE_WRITE_URL>`: URL from the **Yandex Monitoring** -> **Prometheus** -> **Remote write** page.
- `<API KEY>`: API key from the previous step (for example, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`).

You can specify additional parameters as described in the `prometheus` module documentation.

For more details on writing metrics, refer to the Yandex Cloud documentation.

To read metrics in Grafana:

1. Create a service account with the `monitoring.viewer` role.
2. Create an API key for the service account.
3. Create a GrafanaAdditionalDatasource resource:

```
apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <GRAFANA_READ_URL>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_KEY>
```

Where:

- `<GRAFANA_READ_URL>`: URL from the **Yandex Monitoring** -> **Prometheus** -> **Reading Grafana metrics** page.
- `<API KEY>`: API key from the previous step (for example, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`).

You can specify additional parameters as described in the `prometheus` module documentation.

For more details on reading metrics with Grafana, refer to the Yandex Cloud documentation.

Additional resources

- Module `prometheus` custom resources

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in VK Cloud

To manage resources in VK Cloud using Deckhouse Platform, you must connect to the VK Cloud API.

The list of VK Cloud API services required for deployment is available in the Configuration section.

The user credentials required to access the VK Cloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of VK Cloud and using it are available in the VK Cloud documentation.

If you’re using a cloud provider’s implementation of the VK Cloud API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in VK Cloud

List of required OpenStack services

The following VK Cloud services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to order a Load Balancer.

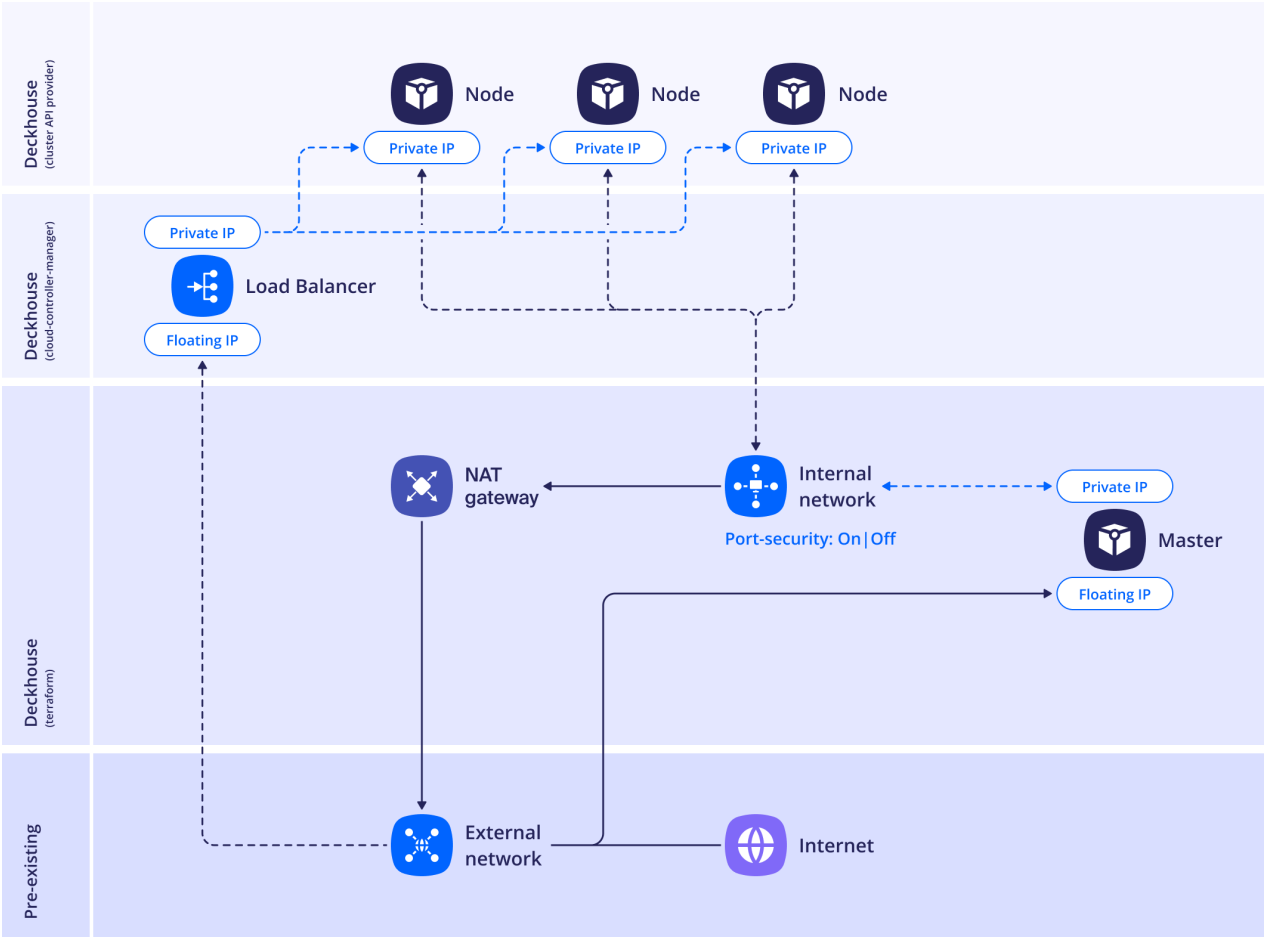
API endpoints and ports are listed in the official documentation.

Layouts

This section describes the possible node placement layouts in VK Cloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
```

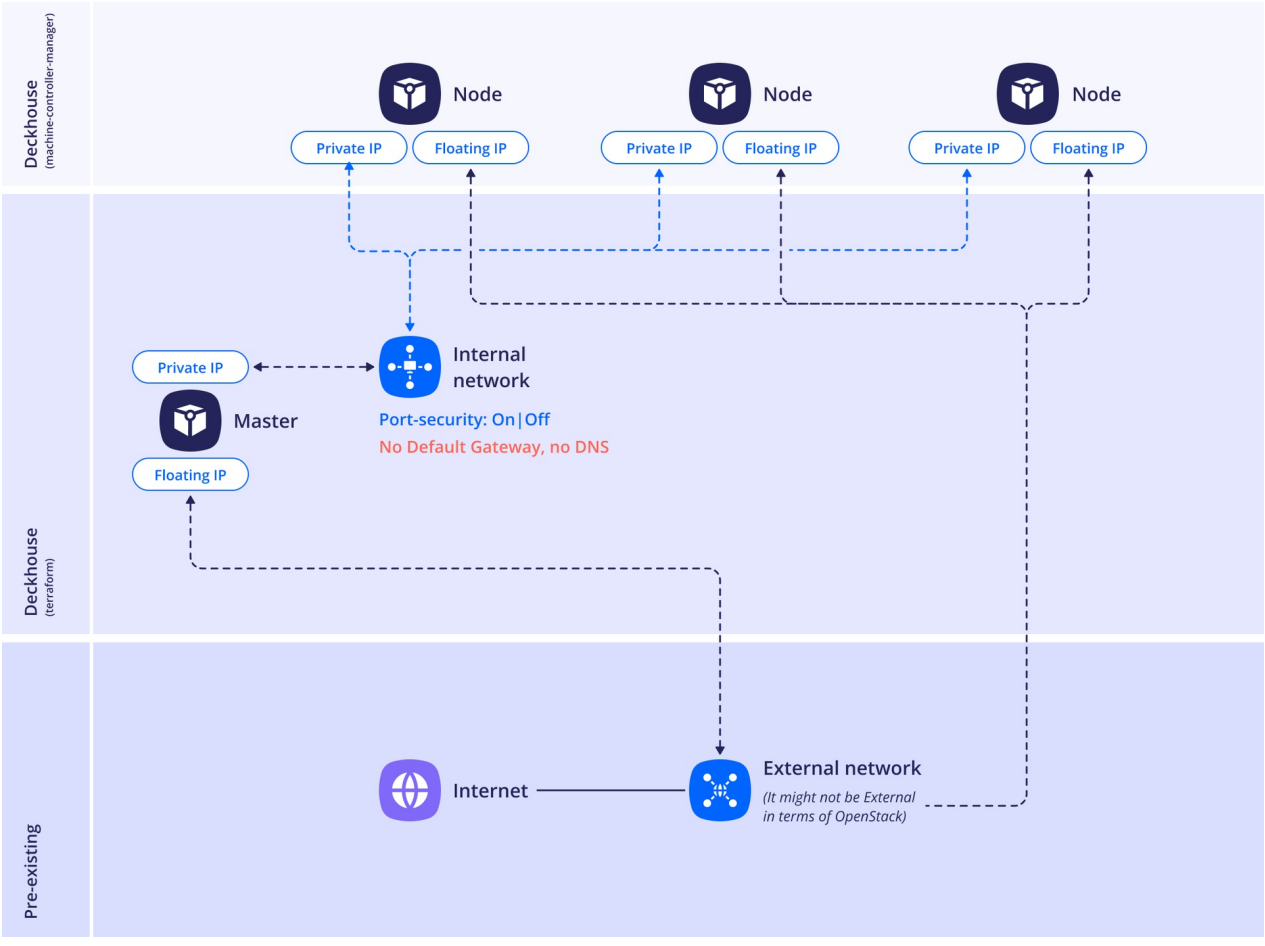
```
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

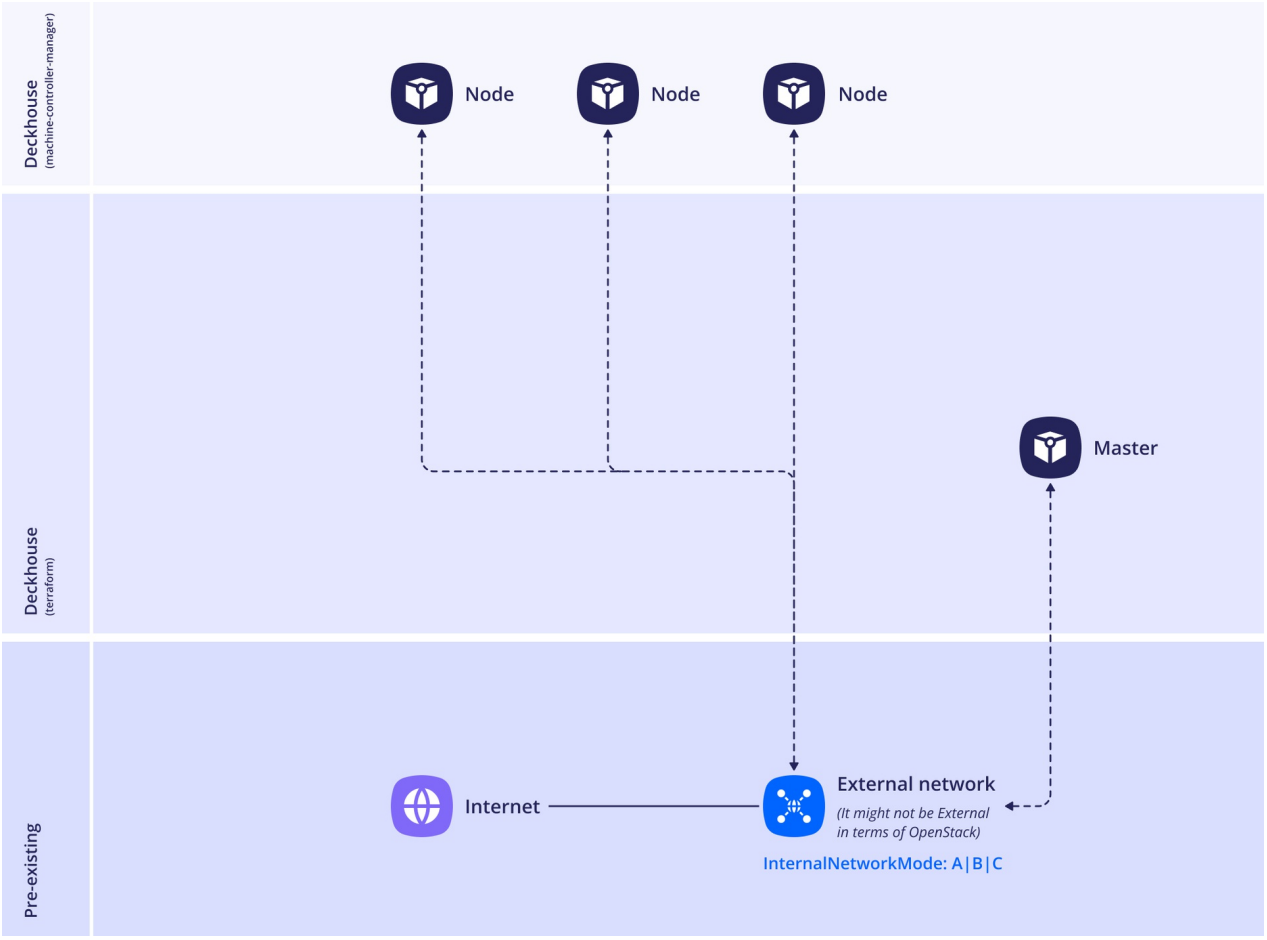
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

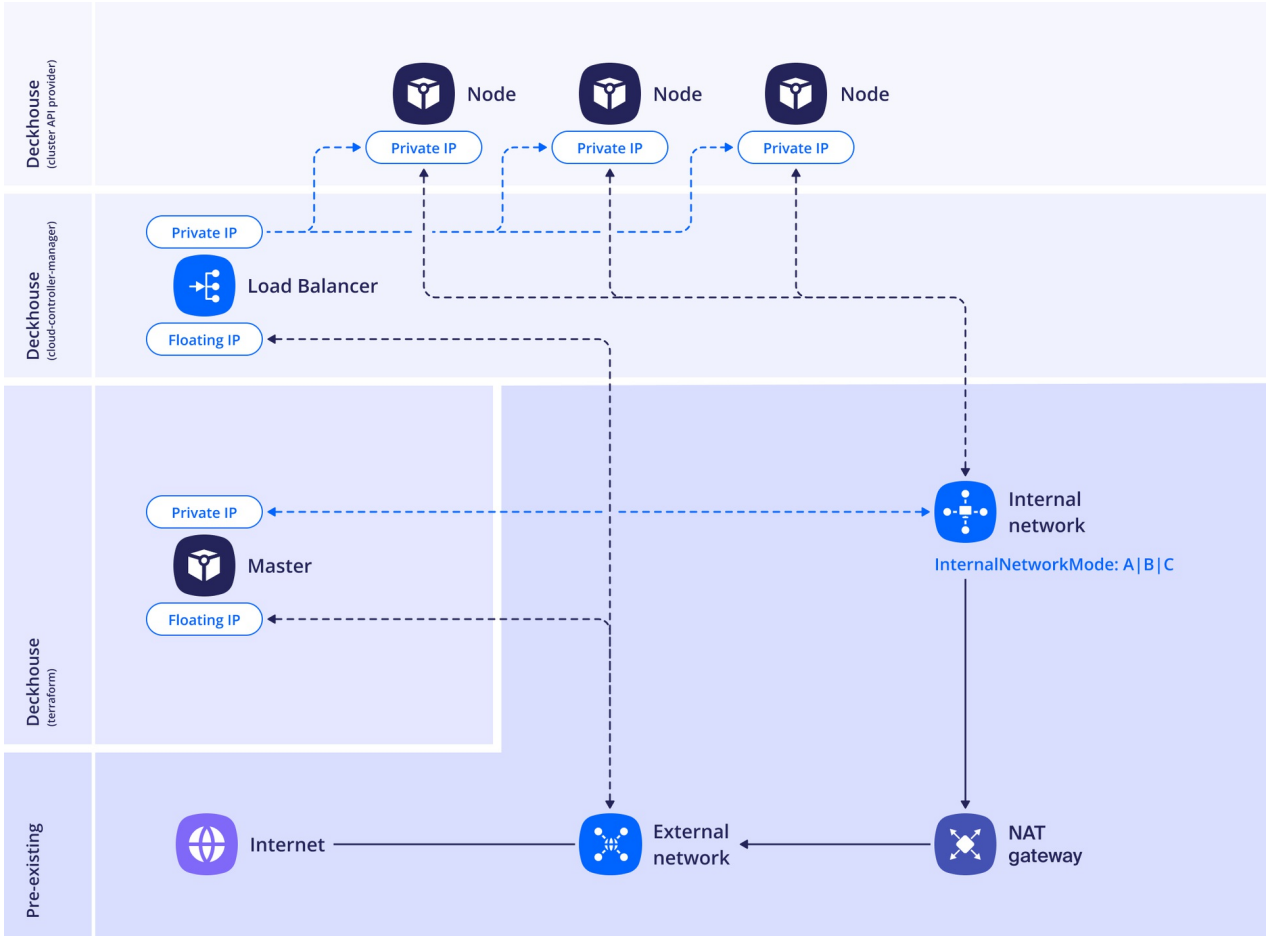
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with VK Cloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in VK Cloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VK Cloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to VK Cloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the VK Cloud API.

   > **i** The method of obtaining the openrc file may vary depending on the VK Cloud provider. If your provider offers the standard interface for VK Cloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if VK Cloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in VK Cloud

A node’s disk can be either local or network-attached. In VK Cloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In VK Cloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:

- If specified in the OS image, it will be used.
- If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new VK Cloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Selectel Cloud

To manage resources in Selectel using Deckhouse Platform, you must connect to the Selectel API.

The list of Selectel API services required for deployment is available in the Configuration section.

The user credentials required to access the Selectel API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of Selectel and using it are available in the Selectel documentation.

If you're using a cloud provider's implementation of the Selectel API, the process for obtaining the openrc file may differ.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in Selectel Cloud

List of required OpenStack services

The following Selectel services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3

Service	API Version
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

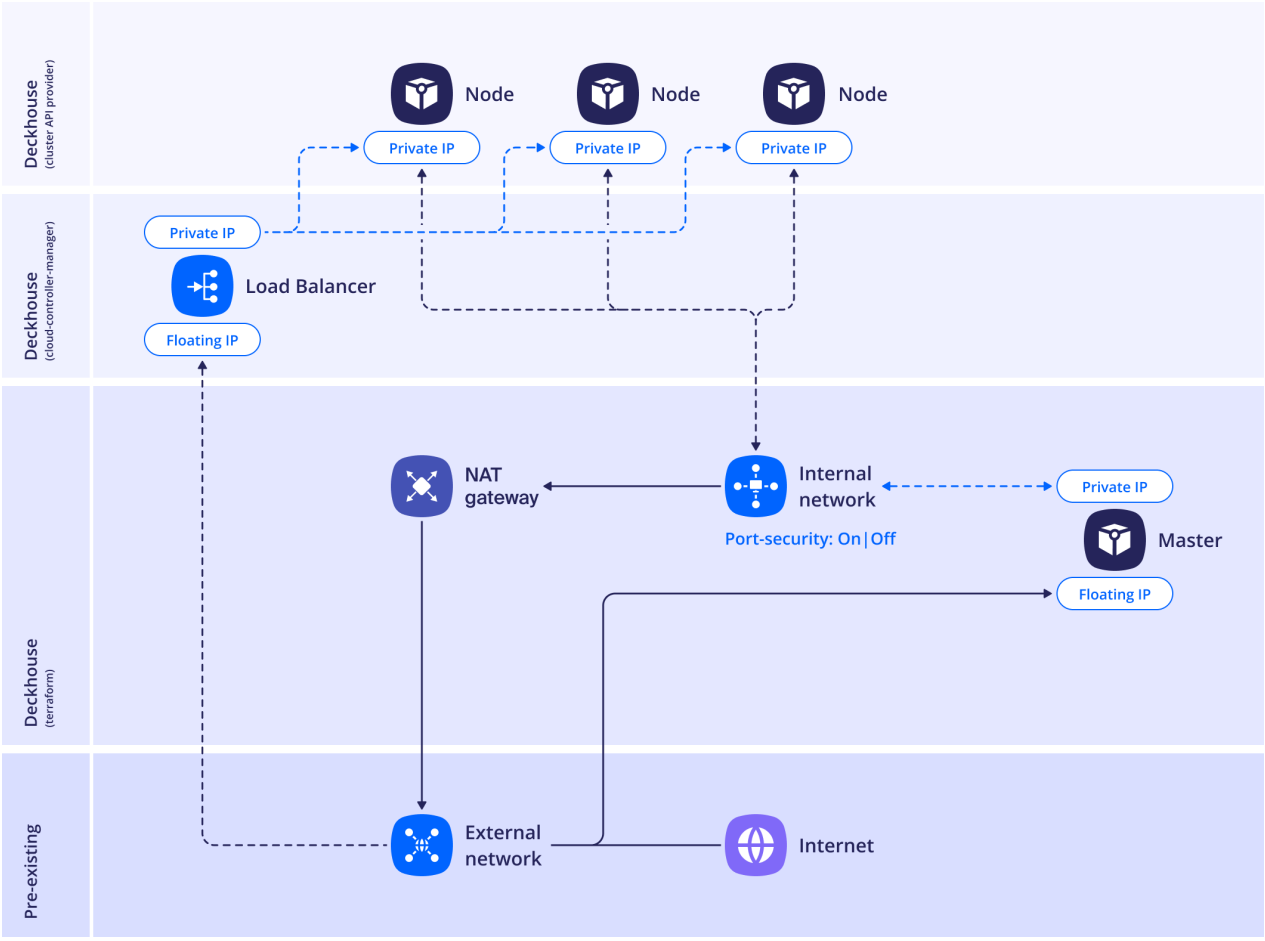
* If you need to order a Load Balancer.

Layouts

This section describes the possible node placement layouts in Selectel infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.

- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
```

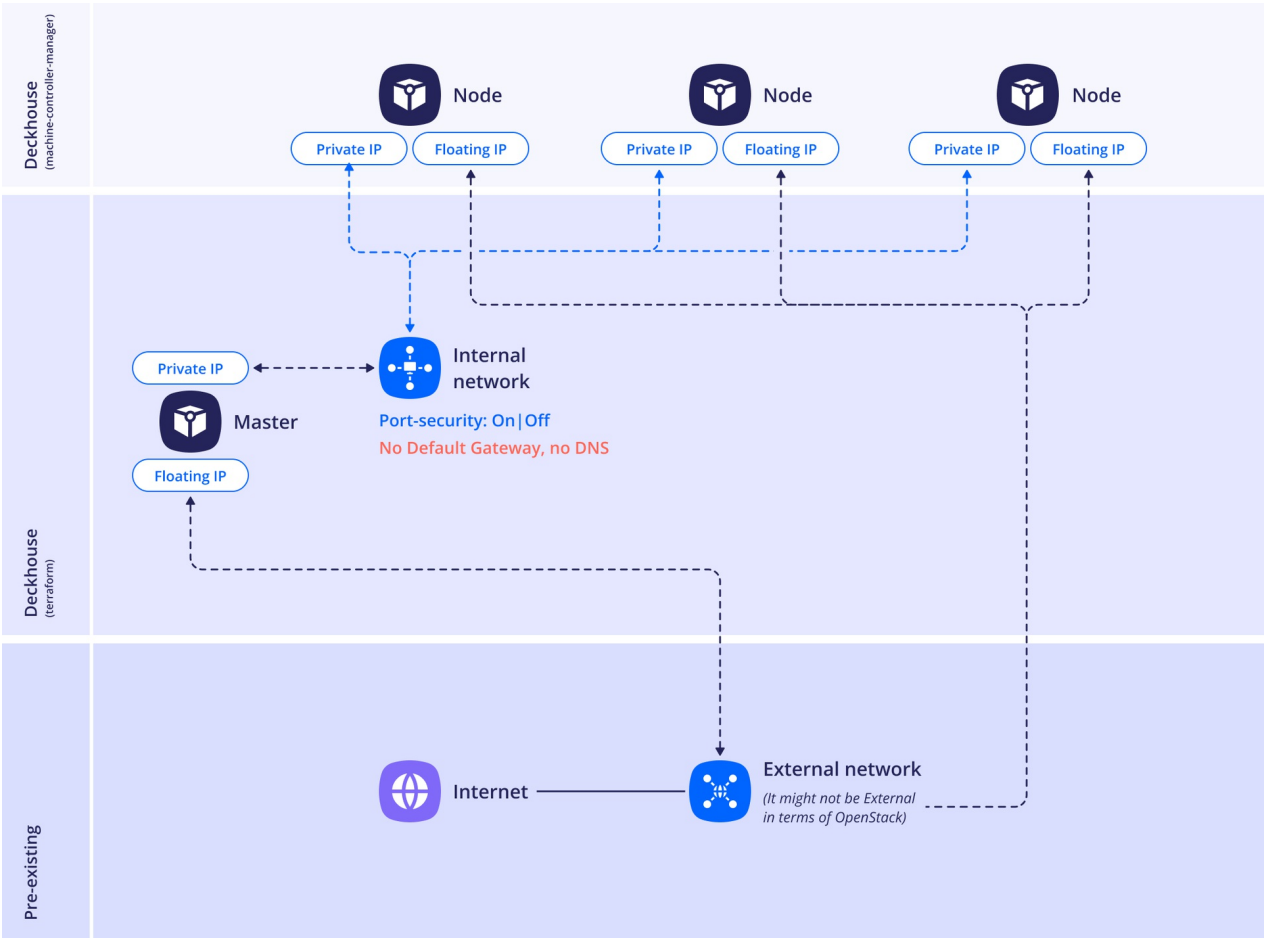
```
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the

OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

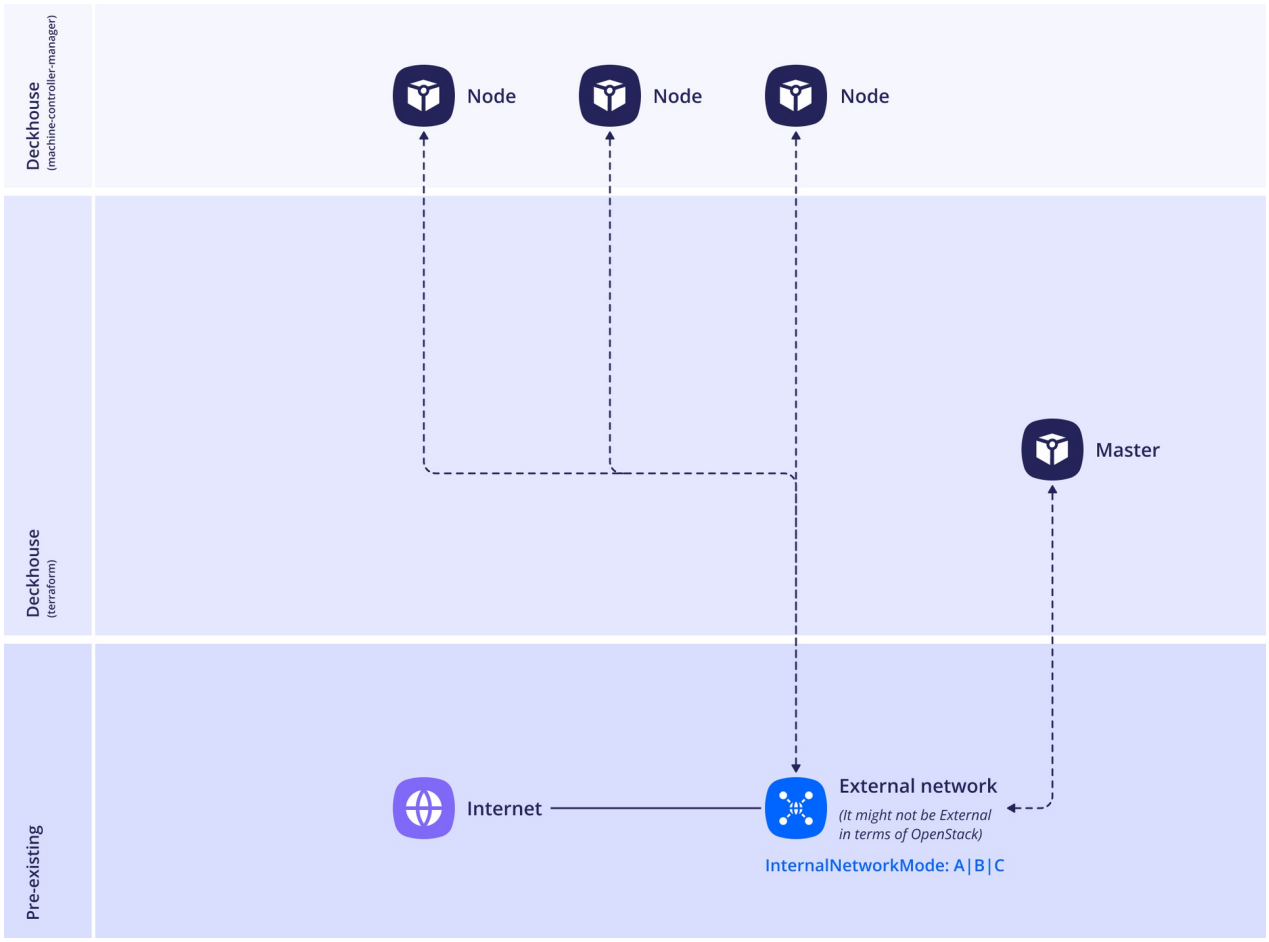
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

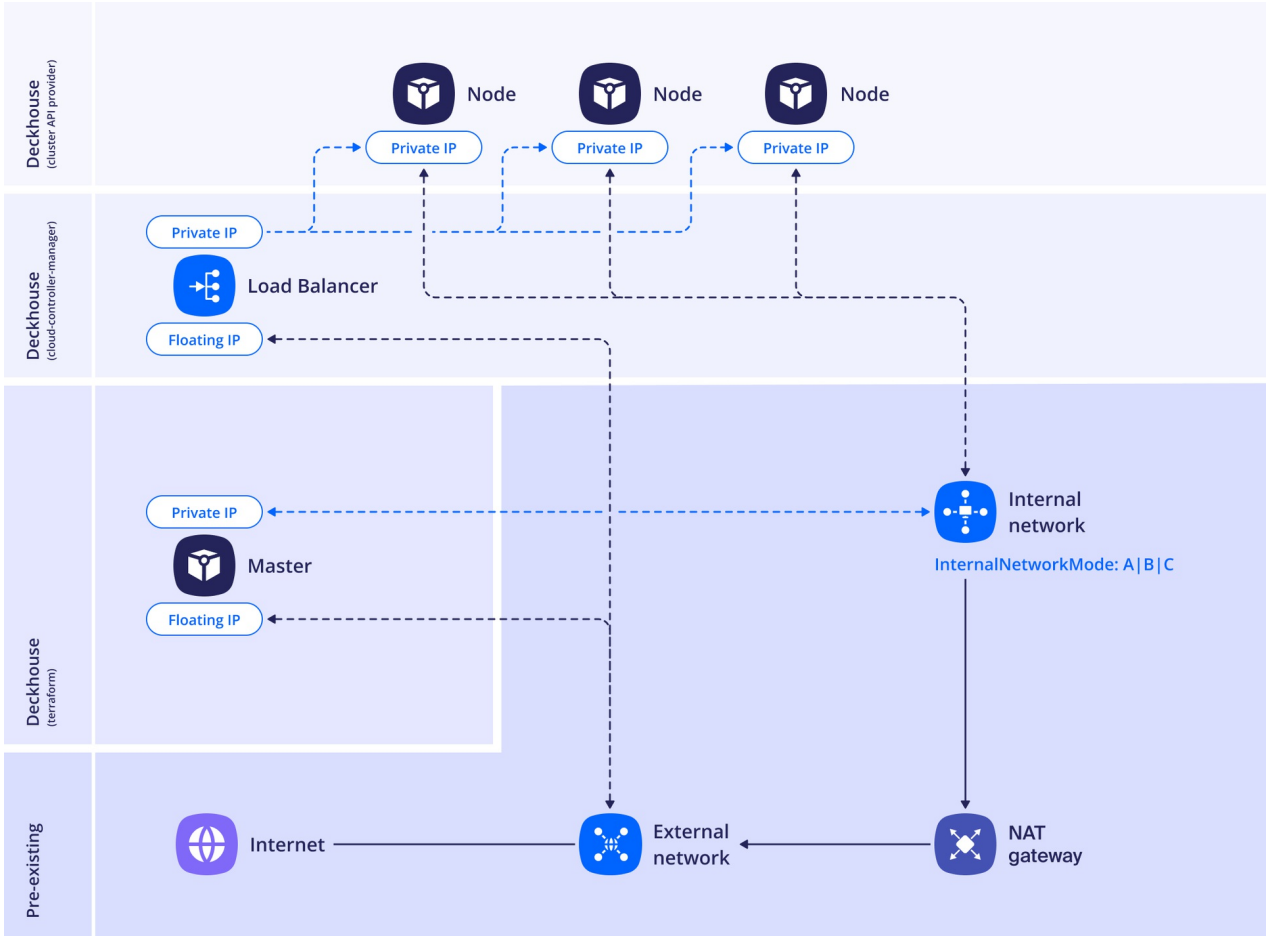
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with Selectel is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in Selectel and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the Selectel cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

How to create preemptible nodes in Selectel

Selectel triggers preemption on the presence of the raw Nova tag `preemptible`. Set the `preemptible` parameter of the OpenStackInstanceClass to `true` — the tag is then attached to the instance automatically.

Example:

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: worker-preempt
spec:
  flavorName: SL1.4-8192
  imageName: Ubuntu 24.04 LTS 64-bit
  rootDiskSize: 30
  preemptible: true
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-preempt
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: worker-preempt
    minPerZone: 2
    maxPerZone: 4
    zones: [ru-3a]
```

Limitations:

- The `preemptible` tag is a Selectel-specific mechanism, and the CAPI provider template emits it only when the cluster's `connection.authURL` points at Selectel (`selcloud.ru` / `selectel`). On other OpenStack providers the tag is silently dropped: the node is still created, but as a regular (non-preemptible) VM.
- Supported only for CloudEphemeral NodeGroups running on the CAPI engine. If an OpenStackInstanceClass with `preemptible: true` is used by a NodeGroup running on the MCM engine, the field is silently ignored — no MCM MachineClass field maps to a raw Nova tag. Nodes are created as regular VMs.
- Both situations above raise the **OpenStackPreemptibleUnsupported** Prometheus alert (with `reason=mcm` or `reason=non-selectel`) so the operator learns about the silent drop instead of discovering it weeks later from an unchanged billing line. The alert description carries the exact commands to fix each case.
- The management engine is selected per NodeGroup. If the same OpenStackInstanceClass with `preemptible: true` is used simultaneously by NodeGroups running on CAPI and MCM, the MCM NodeGroup keeps firing the alert — use separate OpenStackInstanceClass resources for different engines when the preemptibility contract differs.

> ⚠ Changing the `preemptible` parameter in an existing OpenStackInstanceClass changes the OpenStackMachineTemplate and causes all nodes in the corresponding NodeGroup to be recreated.

If a preemptible instance is terminated without a `graceful shutdown`, the corresponding node transitions to the `NotReady` state. After the missing instance is detected, CAPO and MachineHealthCheck initiate the creation of a new Machine and virtual machine. Until recovery is complete, pods on the lost node may still appear in the Kubernetes API with the `Running` status even though they are no longer actually available.

LoadBalancer configuration

> ⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to Selectel

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the Selectel API.

   > **i** The method of obtaining the openrc file may vary depending on the Selectel provider. If your provider offers the standard interface for Selectel, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if Selectel does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in Selectel

A node’s disk can be either local or network-attached. In Selectel terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In Selectel, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new Selectel image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in OVH Cloud

To manage resources in OVHcloud using Deckhouse Platform, you must connect to the OVHcloud API.

The list of OVHcloud API services required for deployment is available in the Configuration section.

The user credentials required to access the OVHcloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OVHcloud and using it are available in the OVHcloud documentation.

If you're using a cloud provider's implementation of the OVHcloud API, the process for obtaining the openrc file may differ.

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in OVH Cloud

List of required OpenStack services

The following OVHcloud services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

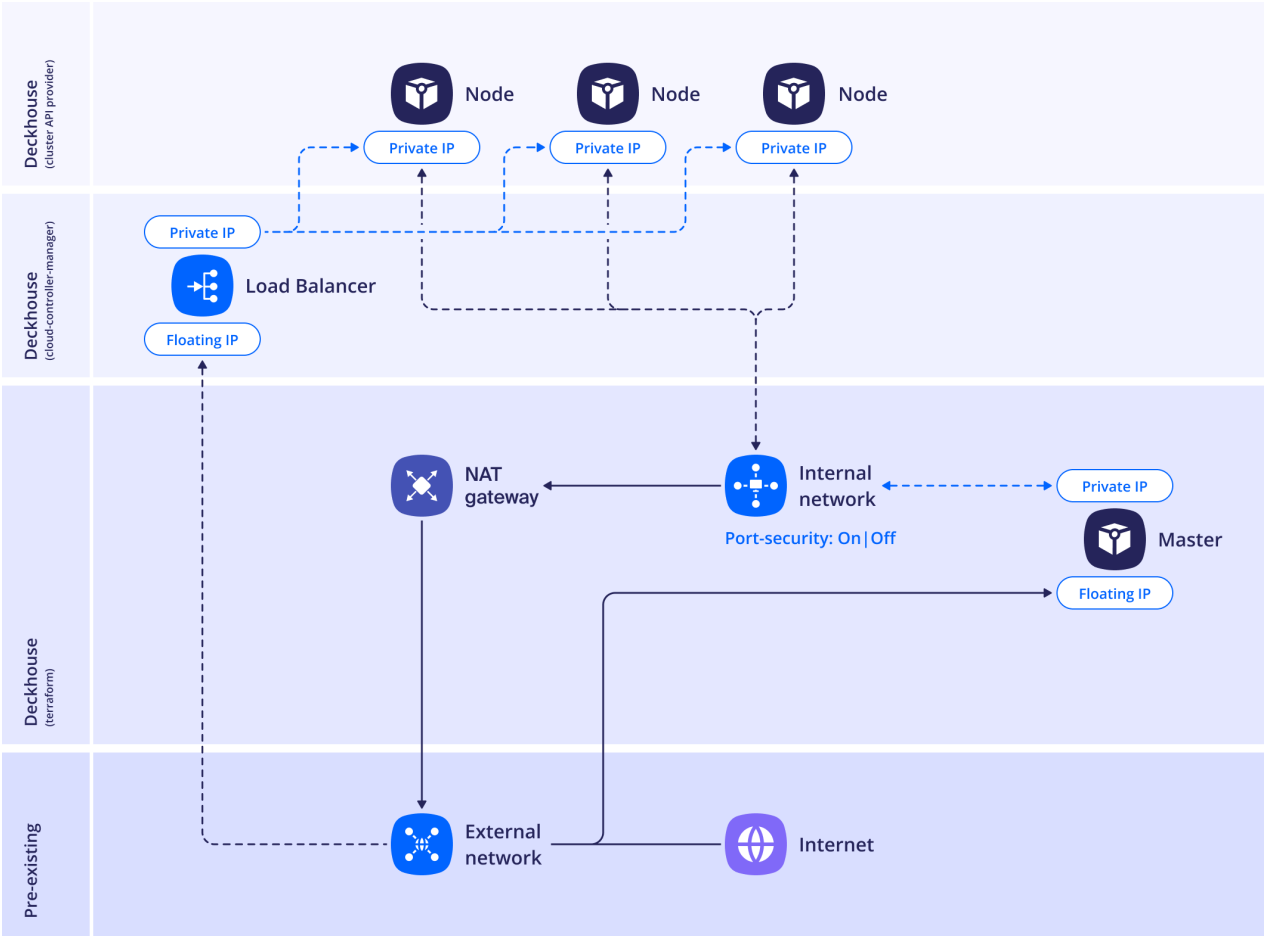
* If you need to order a Load Balancer.

Layouts

This section describes the possible node placement layouts in OVHcloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by

following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
```

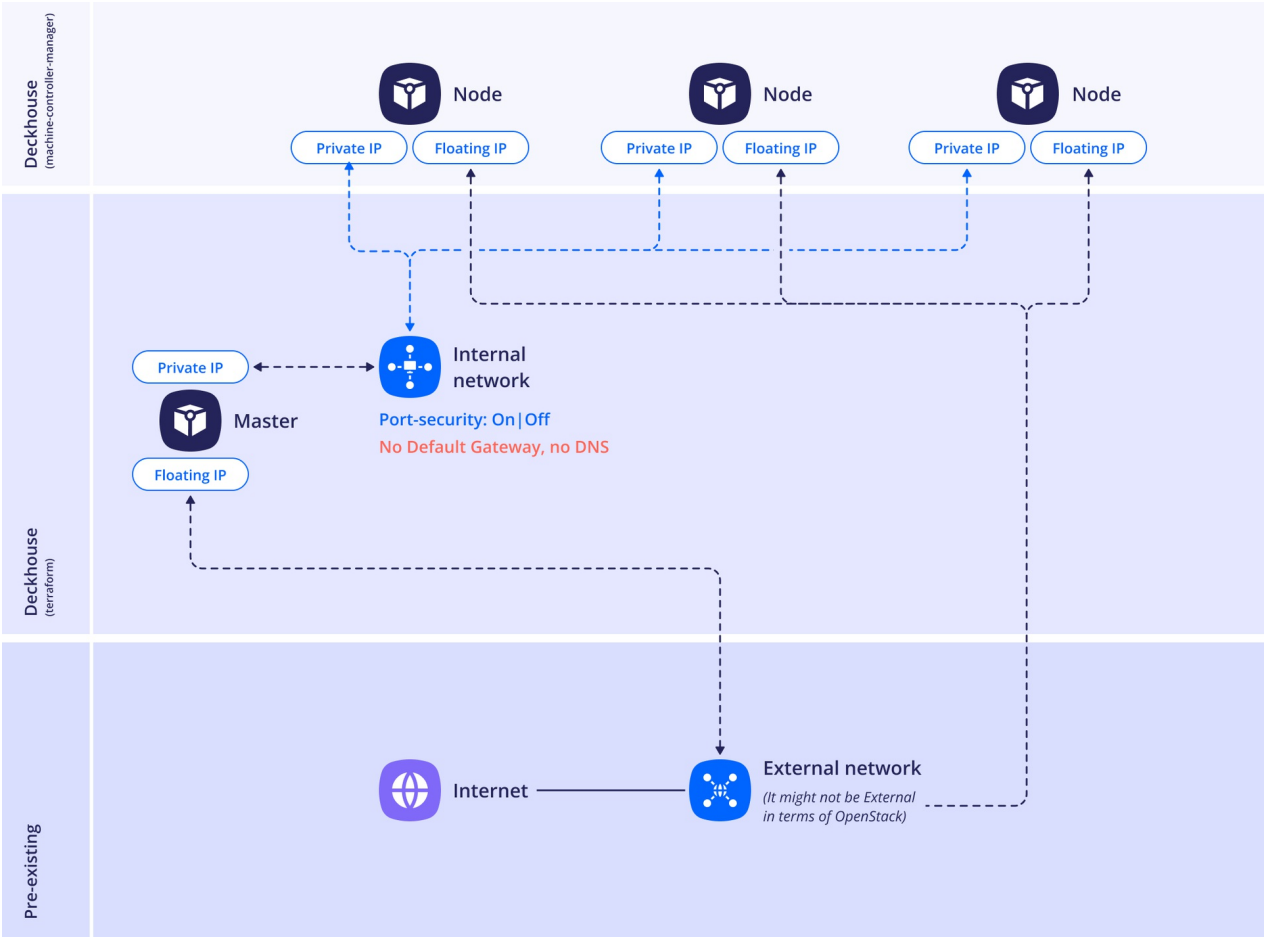
```
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by

following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

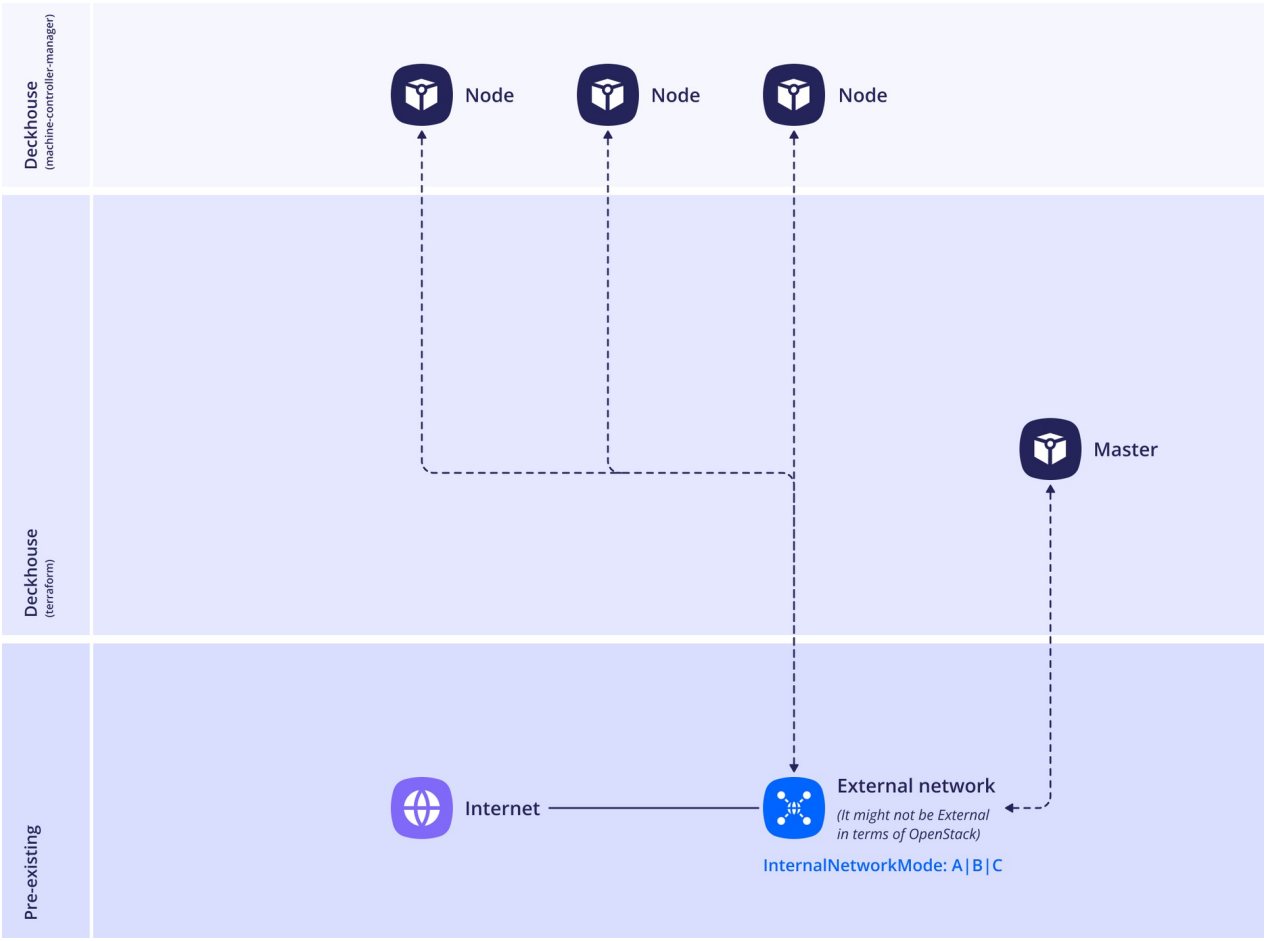
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

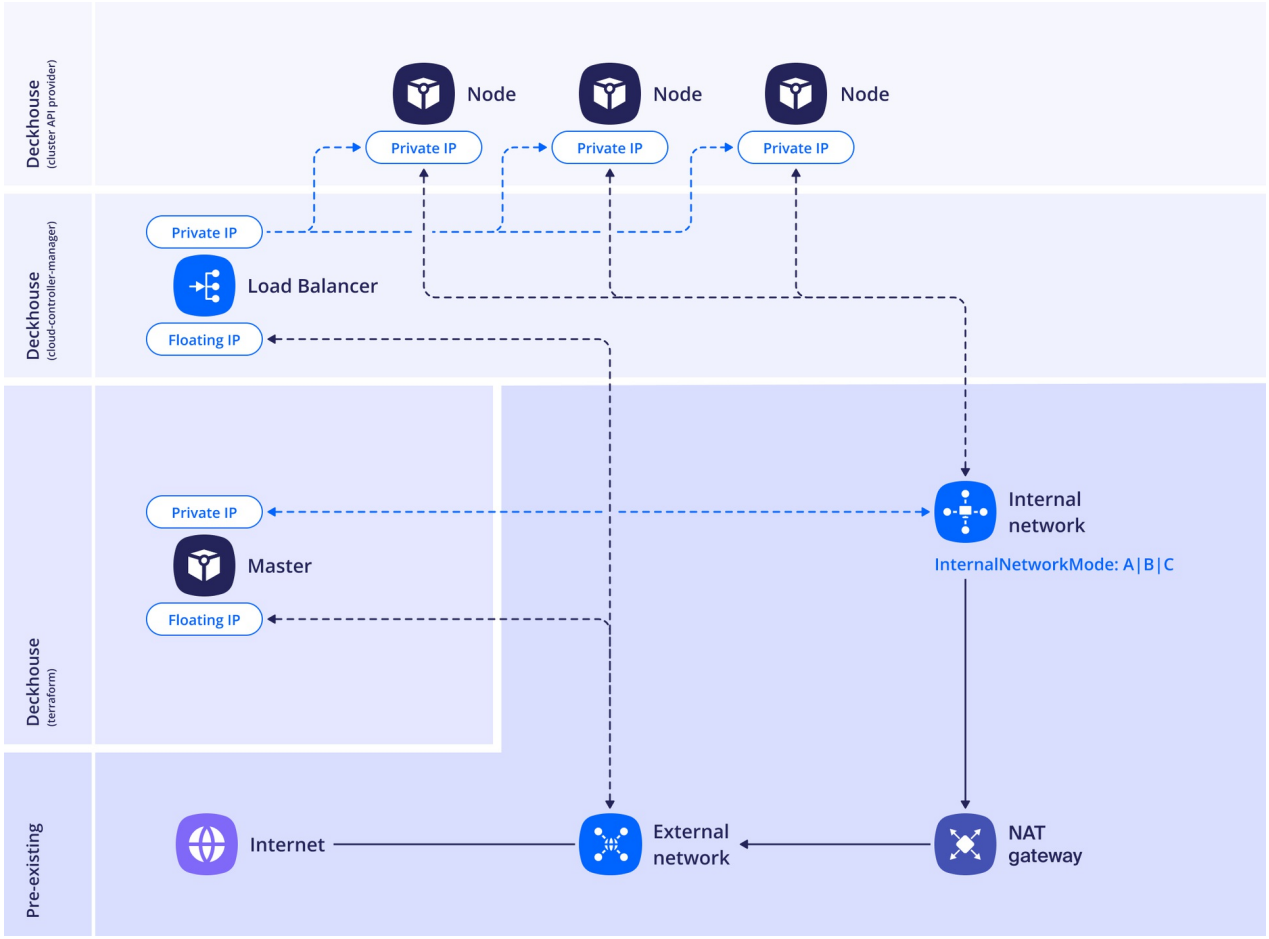
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OVHcloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OVHcloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OVHcloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to OVHcloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OVHcloud API.

   > **i** The method of obtaining the openrc file may vary depending on the OVHcloud provider. If your provider offers the standard interface for OVHcloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OVHcloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OVHcloud

A node’s disk can be either local or network-attached. In OVHcloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OVHcloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:

- If specified in the OS image, it will be used.
- If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OVHcloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in OpenStack

To manage resources in OpenStack using Deckhouse Platform, you must connect to the OpenStack API.

The list of OpenStack API services required for deployment is available in the Configuration section.

The user credentials required to access the OpenStack API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OpenStack and using it are available in the OpenStack documentation.

If you're using a cloud provider's implementation of the OpenStack API, the process for obtaining the openrc file may differ.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in OpenStack

List of required OpenStack services

The following OpenStack services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

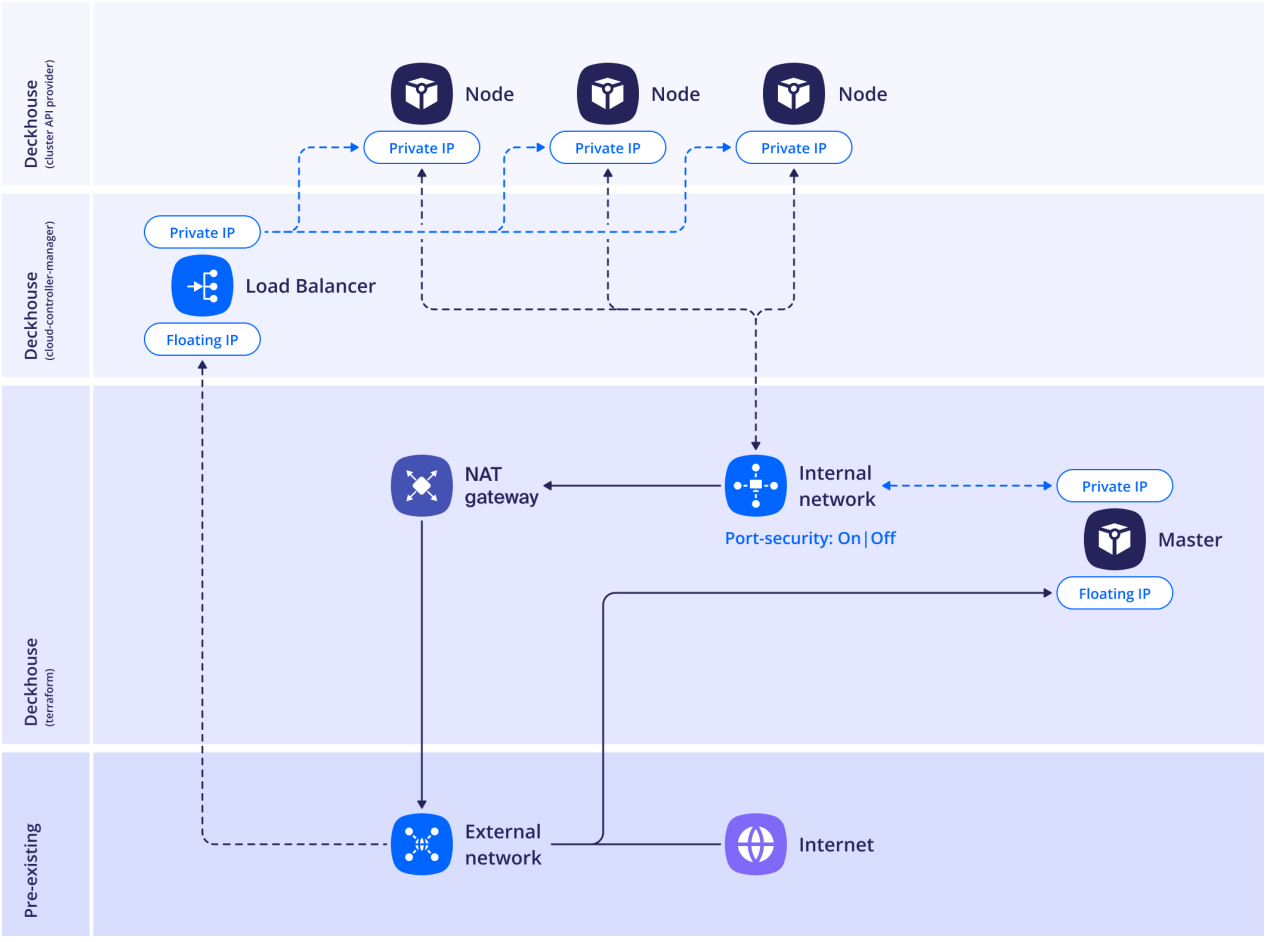
* If you need to order a Load Balancer.

Layouts

This section describes the possible node placement layouts in OpenStack infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



> If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when

a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
```

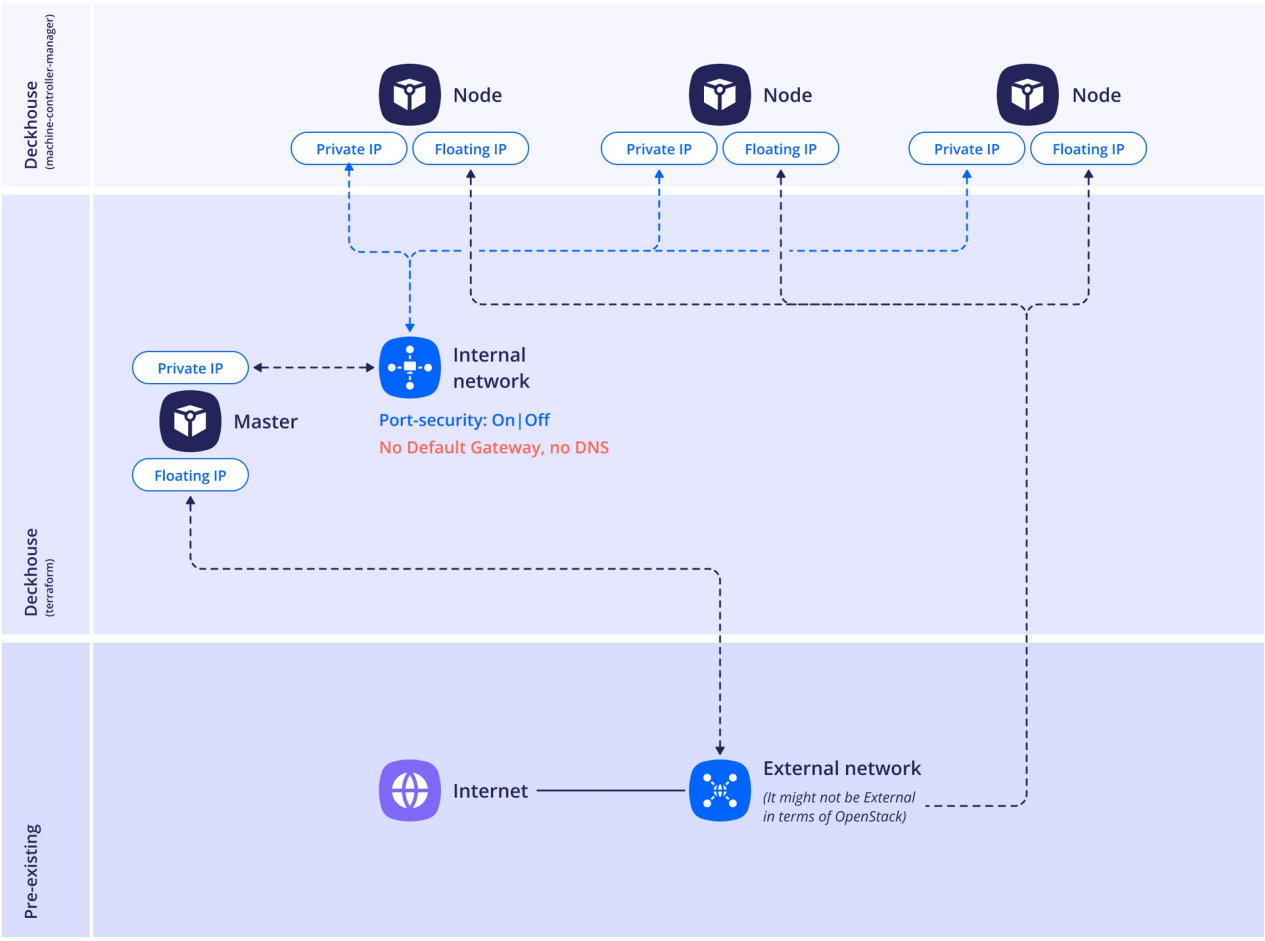
```
    mainNetwork: kube
    additionalNetworks:                  # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when

a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

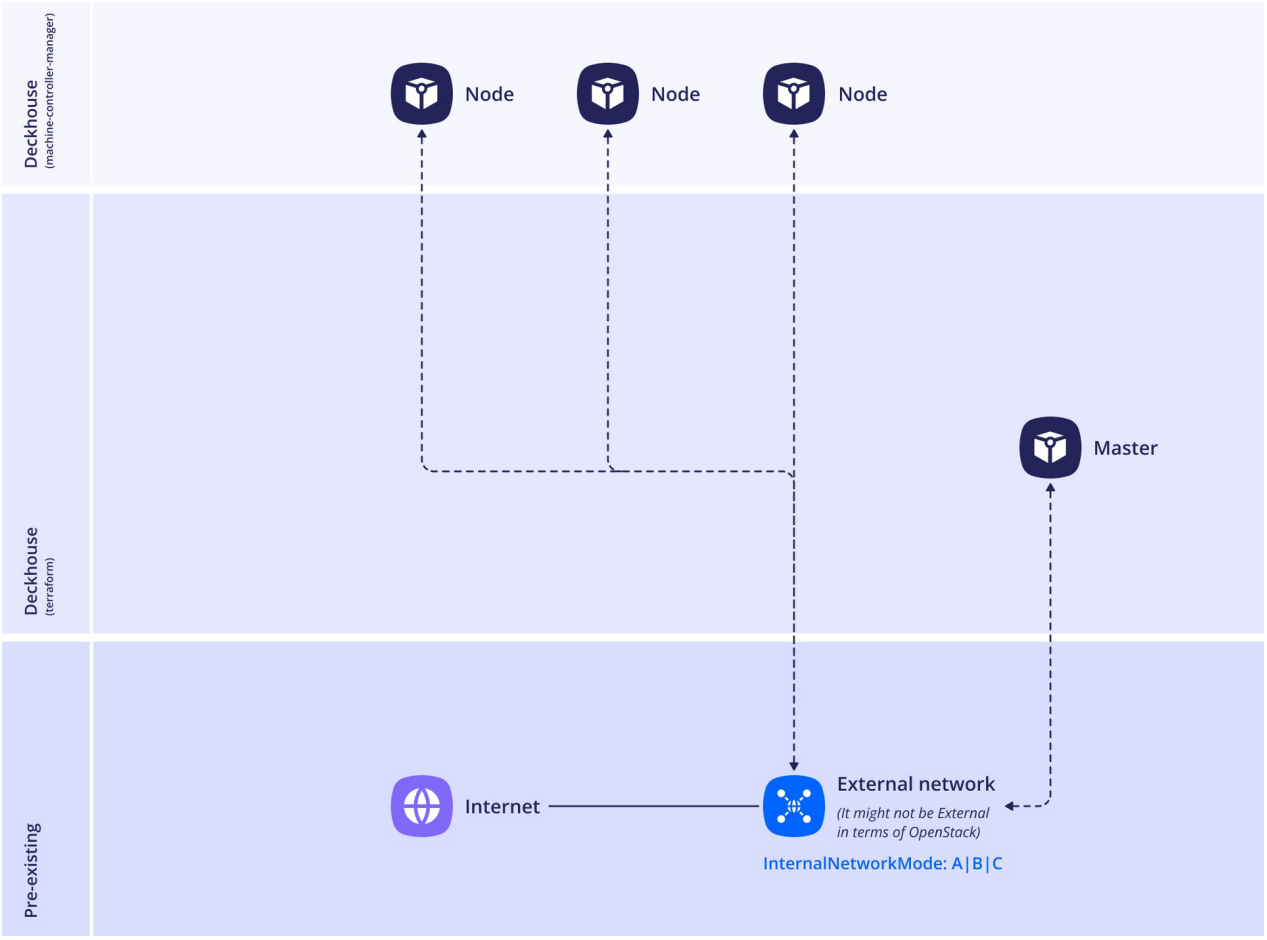
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

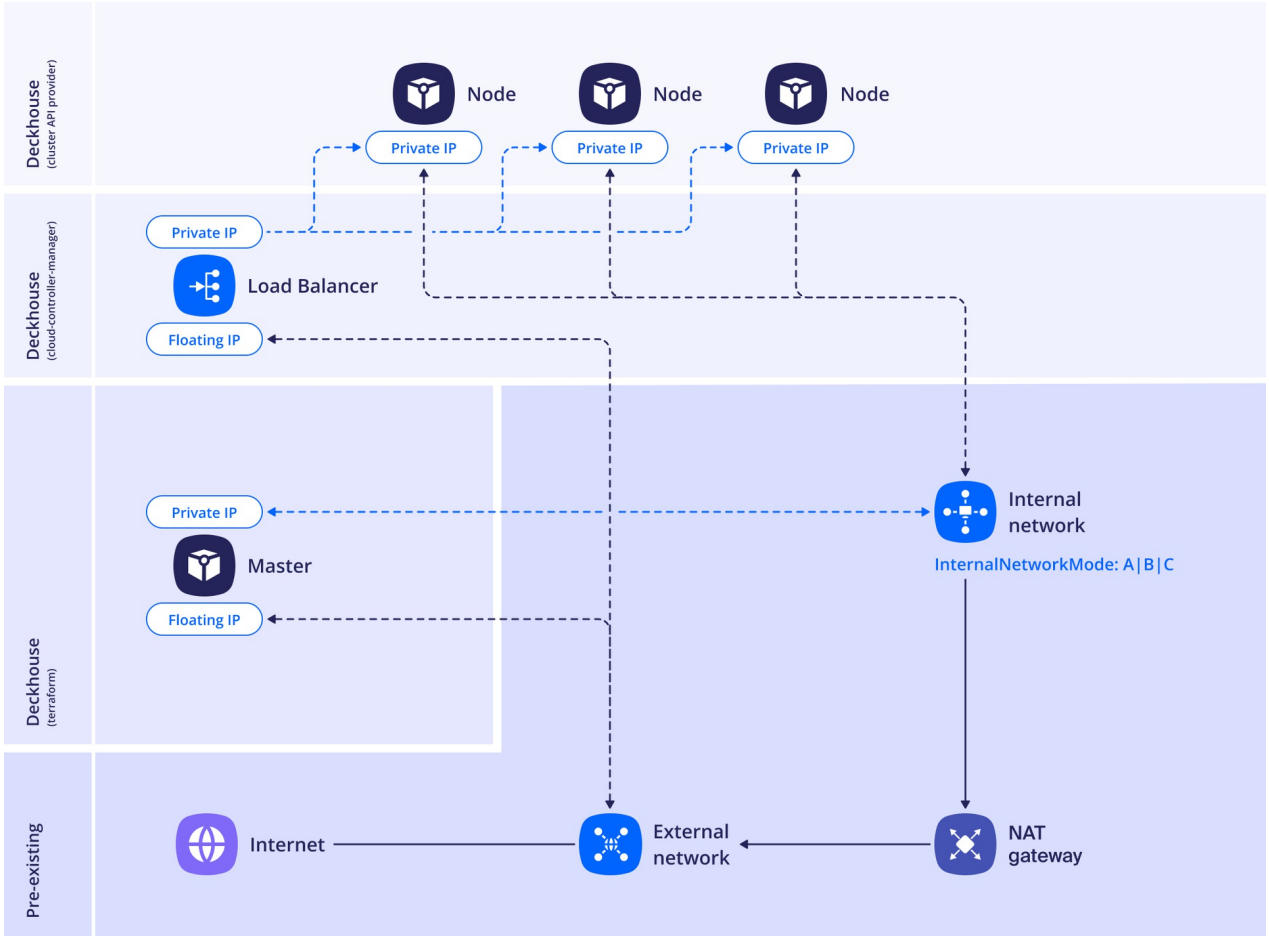
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OpenStack is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OpenStack and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OpenStack cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to OpenStack

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OpenStack API.

   > **i** The method of obtaining the openrc file may vary depending on the OpenStack provider. If your provider offers the standard interface for OpenStack, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OpenStack does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OpenStack

A node’s disk can be either local or network-attached. In OpenStack terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OpenStack, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:

- If specified in the OS image, it will be used.
- If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OpenStack image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Amazon Web Services

Requirements

The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Before you begin, you must prepare the cloud environment and provide access to AWS resources for Deckhouse Platform (DP) components that interact with the AWS API.

The `cloud-init` package must be installed on all virtual machines that will be part of the cluster. After booting up, make sure the following services are running:

- `cloud-config.service`
- `cloud-final.service`
- `cloud-init.service`

These services are required for correct EC2 instance initialization and for integration with DP infrastructure modules such as `cloud-provider-aws` and `machine-controller-manager`.

Accessing the AWS API

To access the AWS API, an IAM user with the required permissions is needed. You must create a JSON policy specification granting access to EC2, ELB, IAM, KMS, and other AWS services.

Example JSON policy with the required permission configuration:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:AttachVolume",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateInternetGateway",
                "ec2:CreateKeyPair",
                "ec2:CreateNATGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateVpc",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteKeyPair",
                "ec2:DeleteNATGateway",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DeleteVpc",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAddressesAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeInstanceTopology",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRegions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVolumesModifications",
                "ec2:DescribeVolumes",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcClassicLink",
                "ec2:DescribeVpcClassicLinkDnsSupport",
                "ec2:DescribeVpcs",
                "ec2:DetachInternetGateway",
                "ec2:DetachVolume",
                "ec2:DisassociateAddress",
                "ec2:DisassociateRouteTable",
```

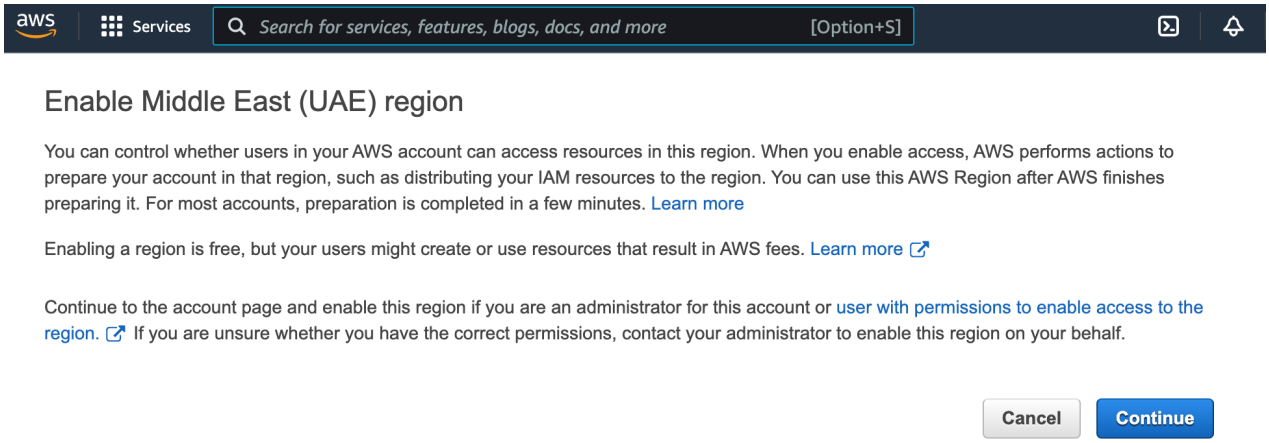
```
                "ec2:ImportKeyPair",
                "ec2:ModifyInstanceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVolume",
                "ec2:ModifyVpcAttribute",
                "ec2:ReleaseAddress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:CreateVpcPeeringConnection",
                "ec2:DeleteVpcPeeringConnection",
                "ec2:AcceptVpcPeeringConnection",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "elasticloadbalancing:AddTags",
                "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
                "elasticloadbalancing:AttachLoadBalancerToSubnets",
                "elasticloadbalancing:ConfigureHealthCheck",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateLoadBalancerListeners",
                "elasticloadbalancing:CreateLoadBalancerPolicy",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteLoadBalancerListeners",
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancerAttributes",
                "elasticloadbalancing:DescribeLoadBalancerPolicies",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DetachLoadBalancerFromSubnets",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:ModifyLoadBalancerAttributes",
                "elasticloadbalancing:ModifyTargetGroup",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets",
                "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
                "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:CreateServiceLinkedRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "kms:DescribeKey",
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

Configuring IAM via the AWS web console

1. Create policies:
   - Go to **Identity and Access Management (IAM)** → **Policies**.
   - Click **Create policy**.
   - Switch to the **JSON** tab and paste the prepared JSON specification.
   - Click **Next: Tags**, then **Next: Review**.
   - Provide a name for the policy, for example, D8CloudProviderAWS.
   - Click **Create policy**.
2. Create a user:
   - Go to **IAM** → **Users**.
   - Click **Add users**.
   - Enter a username, for example, deckhouse.
   - Under **Select AWS credential type**, choose **Access key** → **Programmatic access**.
   - Click **Next: Permissions**.
3. Attach policies to the user:
   - Choose **Attach existing policies directly**.
   - Enter the name of the policy created earlier in the filter field and check the box next to it.
   - Click **Next: Tags**, then **Next: Review**.
   - Click **Create user**.

> Save the generated `Access key ID` and `Secret access key`. You will need these to configure DP.

Ensure that your account (and the created user) has access to the required regions. To verify this, select the region from the dropdown menu in the upper-right corner. If the region switches successfully, access is available. If not, you may see a message like this:



Click **Continue** to enable the region.

Configuring IAM via AWS CLI

You can also configure IAM using the command-line interface. Follow these steps:

1. Save the JSON specification to a file named `policy.json`:

   ```
   cat > policy.json << EOF
   <Policy JSON spec>
   EOF
   ```

2. Create a new policy named `D8CloudProviderAWS` with the `ARN` details using the specification from the `policy.json`:

   ```
   aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
   ```

   Example output:

```
{
   "Policy": {
      "PolicyName": "D8CloudProviderAWS",
      "PolicyId": "AAA",
      "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
      "Path": "/",
      "DefaultVersionId": "v1",
      "AttachmentCount": 0,
      "PermissionsBoundaryUsageCount": 0,
      "IsAttachable": true,
      "CreateDate": "2020-08-27T02:52:06+00:00",
      "UpdateDate": "2020-08-27T02:52:06+00:00"
   }
}
```


3. Create a new user:

   ```
   aws iam create-user --user-name deckhouse
   ```

   Example output:


   ```
   {
      "User": {
         "Path": "/",
         "UserName": "deckhouse",
         "UserId": "AAAXXX",
         "Arn": "arn:aws:iam::123:user/deckhouse",
         "CreateDate": "2020-08-27T03:05:42+00:00"
      }
   }
   ```


4. Grant access to the API and save `AccessKeyId` and `SecretAccessKey`:

   ```
   aws iam create-access-key --user-name deckhouse
   ```

   Example output:


   ```
   {
      "AccessKey": {
         "UserName": "deckhouse",
         "AccessKeyId": "XXXYYY",
         "Status": "Active",
         "SecretAccessKey": "ZZZzzz",
         "CreateDate": "2020-08-27T03:06:22+00:00"
      }
   }
   ```


5. Attach the policy to the user combining `User` and `Policy`:

   ```
   aws iam attach-user-policy --user-name username --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
   ```

Configuring IAM via Terraform

Example Terraform resource configuration:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name       = "D8CloudProviderAWS"
  path       = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON policy specification>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user      = aws_iam_user.user.name
  policy_arn = aws_iam_policy.policy.arn
}
```

Using a custom IAM role for nodes (iamNodeRole)

The `iamNodeRole` parameter in the AWSClusterConfiguration resource lets you override the default IAM role that DP assigns to all EC2 instances of the cluster nodes.

By default, DP creates and assigns an IAM role named `<prefix>-node`, where `<prefix>` is the value of the `global` ModuleConfig `prefix` parameter. This role includes the minimum permissions required for node operation.

You can specify the IAM role name in the `iamNodeRole` parameter if you want to:

- Use a pre-created IAM role managed manually.
- Grant additional permissions to nodes (for example, access to specific AWS services).

> **i** The specified role must include all permissions required by DP. Without them, the nodes will not function properly. In particular, it must include the same permissions as the default `<prefix>-node` role, even if they are assigned manually.

For more information on IAM roles for AWS EC2, refer to the AWS documentation.

Additional resources

- Module `cloud-provider-aws` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

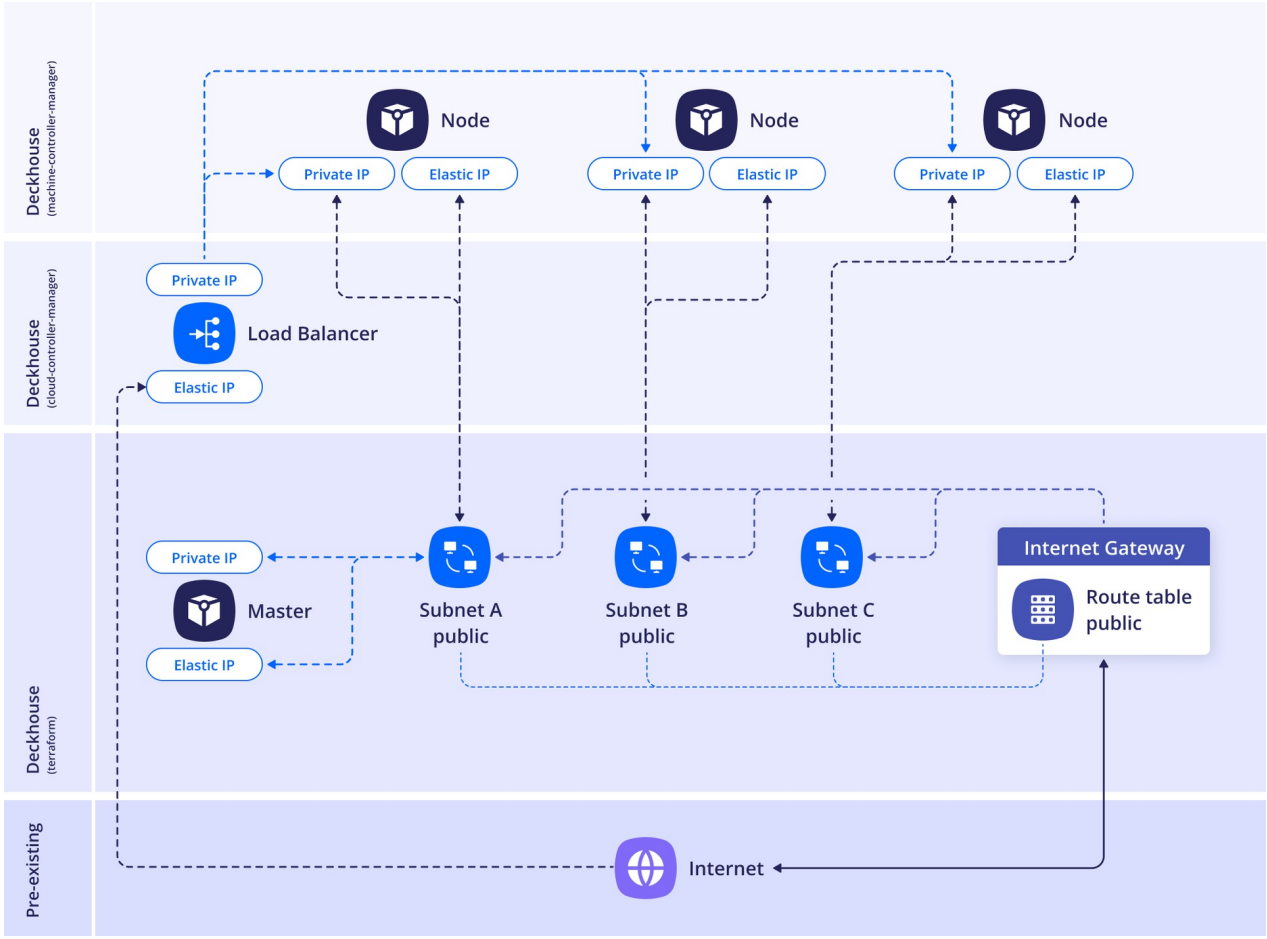
Layouts and configuration in Amazon Web Services

This section describes cluster layouts in AWS infrastructure and their related parameters. The selected layout affects how public IP addresses are assigned, NAT usage, and how nodes are accessed.

WithoutNAT

This is the recommended layout.

Each node is assigned a public IP address (Elastic IP), and no NAT Gateway is used. This layout allows direct access to nodes via public IP addresses and simplifies outgoing traffic routing.



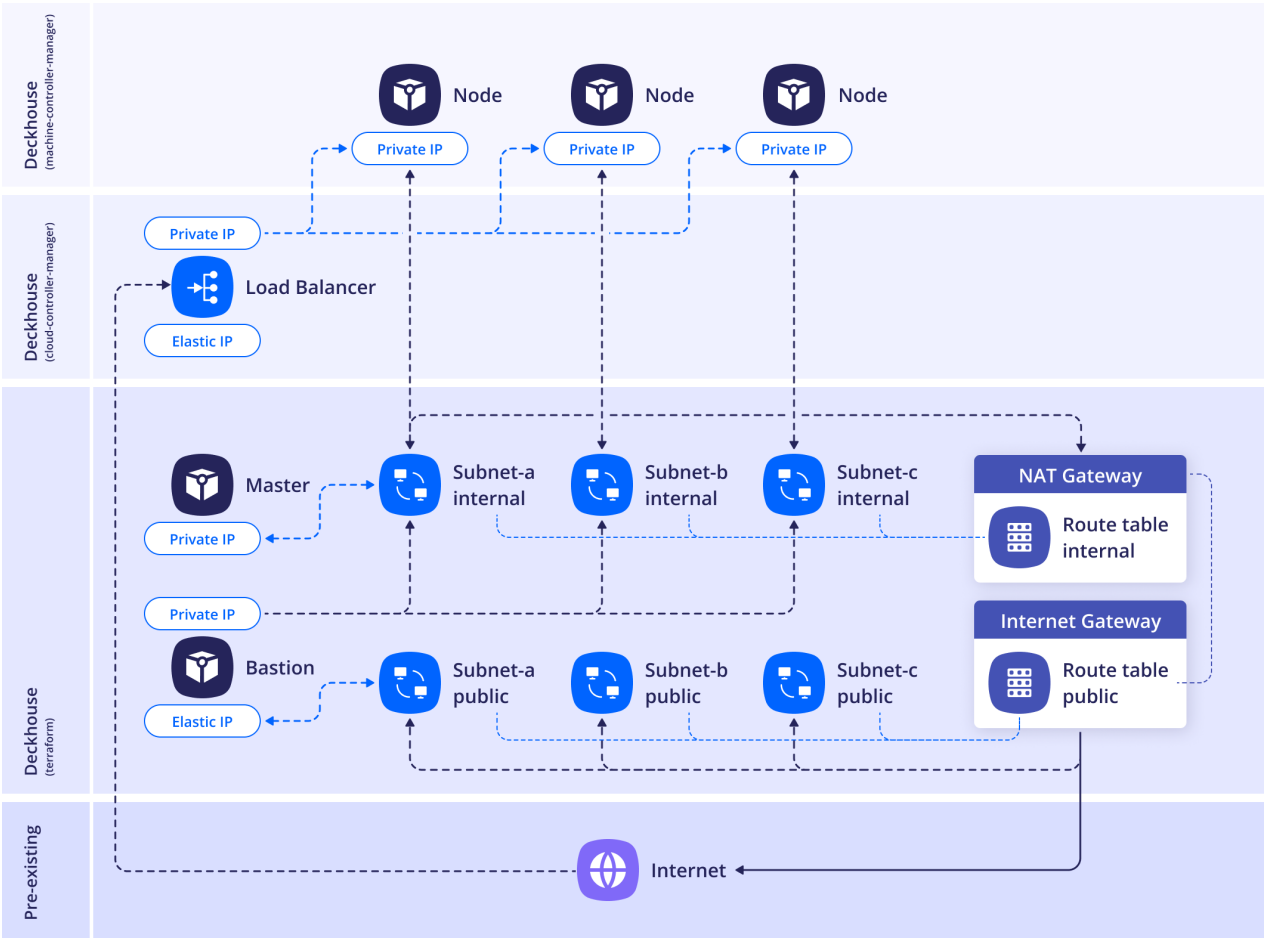
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  # Number of master nodes.
  # If you specify more than one master node, an etcd cluster will be created automatically.
  replicas: 1
  instanceClass:
    # Instance type to use.
    instanceType: m5.xlarge
    # ID of the Amazon Machine Image (AMI).
    # AMI catalog can be found in the AWS Console: EC2 → AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Disk size for the master node VM.
    diskSizeGb: 30
    # Disk type to use for the master node VM.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: torpedo
```

WithNAT

⚠ In this layout, the NAT Gateway is always created in zone `a`. If cluster nodes are placed in other zones, a failure in zone a may cause the cluster to become unavailable. A bastion host is required to connect to the nodes.

In this layout, a NAT Gateway is used for accessing the internet, and nodes do not receive public IP addresses. Access to nodes is only possible through a bastion host, which is placed in a separate subnet.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Number of master nodes.
  # If you specify more than one master node, an etcd cluster will be created automatically.
  replicas: 1
  instanceClass:
    # Instance type to use.
    instanceType: m5.xlarge
    # ID of the Amazon Machine Image (AMI).
    # AMI catalog can be found in the AWS Console: EC2 → AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Disk size for the master node VM.
    diskSizeGb: 30
    # Disk type to use for the master node VM.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  team: torpedo
```

Defining AWSClusterConfiguration

The AWSClusterConfiguration resource describes the cluster settings and is used by Deckhouse Platform (DP) to:

- Define the layout and network CIDRs.
- Configure master and worker nodes.
- Specify AWS API connection parameters (such as access keys and region).
- Define common and custom tags.
- Configure the bastion host (for the WithNAT layout).

Required fields:

- `apiVersion`: Must be set to `deckhouse.io/v1`.
- `kind`: Always AWSClusterConfiguration.

Example resource header:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
```

To edit this resource in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

After editing, apply the changes with the following command:

```
dhctl converge
```

Internal addressing and subnets

The `nodeNetworkCIDR` parameter defines the address range to be split among availability zones. This range must match or be a subset of `vpcNetworkCIDR`. Subnets are automatically created based on the number of zones in the region.

Example:

```
nodeNetworkCIDR: 10.241.1.0/20
vpcNetworkCIDR: 10.241.0.0/16
```

Security groups

AWS security groups are used to manage incoming and outgoing traffic to virtual machines. In DP, they can be used to:

- Allow access to cluster nodes from other subnets.
- Open access to applications running on static nodes.
- Restrict or allow access to external resources based on security policies.

When the `disableDefaultSecurityGroup: false` parameter is set, DP creates the following default security groups when a cluster is created:

- `<CLUSTER_PREFIX>-node`, assigned to cluster nodes:
  - Allows any outgoing traffic to `0.0.0.0/0`.
  - Allows any incoming traffic from the `<CLUSTER_PREFIX>-loadbalancer` group.
  - Allows any incoming traffic from nodes in the same `<CLUSTER_PREFIX>-node` group.
  - Allows incoming traffic over the ICMP protocol from the CIDRs listed in `publicNetworkAllowList` (default `0.0.0.0/0`).
- `<CLUSTER_PREFIX>-loadbalancer`, used by load balancers:
  - Allows any incoming traffic from the CIDRs listed in `publicNetworkAllowList`.
  - Allows any outgoing traffic to the `<CLUSTER_PREFIX>-node` group.
- `<CLUSTER_PREFIX>-ssh-accessible`, created if `sshAllowList` is set. It allows incoming traffic over the TCP protocol on port `22` from the listed CIDRs (default `0.0.0.0/0`). It is assigned to master nodes or to the bastion host in the WithNAT layout.

In addition to the default groups, you can attach custom security groups created in the cloud beforehand.

You can assign additional security groups in the following cases:

Node type	Where to configure
Master nodes	In the `masterNodeGroup.instanceClass.additionalSecurityGroups` field of the AWSClusterConfiguration resource
Static worker nodes	In the `nodeGroups[].instanceClass.additionalSecurityGroups` field of the same resource
Ephemeral nodes	In the `spec.additionalSecurityGroups` field of the AWSInstanceClass object

In all cases, the `additionalSecurityGroups` parameter must be an array of strings (AWS security group IDs or names).

If `disableDefaultSecurityGroup: true` is set, the default security groups will not be created.

When using `disableDefaultSecurityGroup: true`, you must manually create all required security groups to allow access to cluster nodes. Additionally, you must explicitly specify them in the following parameters:

- `additionalSecurityGroups` in the `masterNodeGroup` section of the AWSClusterConfiguration resource;
- `additionalSecurityGroups` in the AWSInstanceClass resource;
- `additionalSecurityGroups` in the `nodeGroups.instanceClass` section.

To configure the security groups used by load balancers, specify them using the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

Setting up VPC peering

The following example shows how to set up VPC peering between two sample VPCs: `vpc-a` and `vpc-b`.

⚠ The IPv4 CIDR blocks of both VPCs must be different.

To configure peering, follow these steps:

1. Go to the region where `vpc-a` is working.
2. Navigate to VPC → VPC Peering Connections → Create Peering Connection, then configure the peering:
   - Name: `vpc-a-vpc-b`.
   - Fill in “Local” and “Another VPC” sections.
3. Go to the region where `vpc-b` is working.
4. Navigate to VPC → VPC Peering Connections.
5. Select the created connection and click **Accept Request**.
6. In `vpc-a`, add routes to the `vpc-b` CIDR in all route tables via the peering connection.
7. In `vpc-b`, add routes to the `vpc-a` CIDR in all route tables via the peering connection.

Configuring access via bastion host

To access nodes in private subnets, use the `withNAT.bastionInstance` parameter in AWSClusterConfiguration. The

bastion host is provisioned with the infrastructure according to the `instanceClass` settings.

The following scenarios are supported:

- The bastion host already exists in an external VPC:
  1. Run the base infrastructure bootstrap: `dhctl bootstrap-phase base-infra`.
  2. Set up VPC peering between the external and new VPC.
  3. Continue installation using the bastion host: `dhctl bootstrap --ssh-bastion...`.
- The bastion host is required in the new VPC:
  1. Run the base infrastructure bootstrap: `dhctl bootstrap-phase base-infra`.
  2. Manually launch the bastion host in the `<CLUSTER PREFIX>-public-0` subnet.
  3. Continue installation using the bastion host: `dhctl bootstrap --ssh-bastion...`.

Creating a cluster in a new VPC with an existing bastion host

1. Bootstrap the cluster's base infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Set up VPC peering as described above.
3. Continue cluster installation. When prompted about the Terraform cache, answer with `y`:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Creating a cluster in a new VPC and deploying a bastion host

1. Bootstrap the cluster's base infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Manually launch a bastion host in the `<CLUSTER_PREFIX>-public-0` subnet.
3. Continue cluster installation. When prompted about the Terraform cache, answer with `y`:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Using an existing VPC (existingVPCID)

The `existingVPCID` parameter in the AWSClusterConfiguration resource lets you use an existing VPC for DP cluster deployment instead of automatically creating a new one.

This may be useful when:

- Your AWS infrastructure is already partially deployed.
- You need to integrate with other services or resources in the same VPC.
- Security policies or architectural guidelines prevent automatic VPC creation.

> ⚠ If the existing VPC already has an Internet Gateway, the base infrastructure bootstrap will fail. Reusing an existing Internet Gateway is not supported in the current DP version.

Compatibility with other parameters:

- If `existingVPCID` is specified, do not specify `vpcNetworkCIDR` as they are mutually exclusive.
- The `nodeNetworkCIDR` parameter can (and should) be specified. It must be a subset of the existing VPC.

Preemptible nodes

Preemptible virtual machines in AWS are called Spot Instances. To make a node group order them, set the `spot` parameter of the AWSInstanceClass resource to `true`.

When an EC2 Instance Rebalance Recommendation signals an elevated risk of Spot Instance interruption, the `node-termination-handler` component automatically applies the `aws-node-termination-handler/rebalance-recommendation` taint with `effect: NoSchedule` to the CloudEphemeral node. This taint prevents the scheduler from placing new workloads on the node unless they have a matching toleration.

The `aws-node-termination-handler/spot-itn` taint indicates a notice of an upcoming Spot Instance interruption.

If either taint is present, the `node-manager` module cordons and drains the node, then deletes it. If the node count in a zone falls below the `minPerZone` parameter value, the group provisions new nodes.

The recommendation allows node decommissioning to start before an interruption notice arrives. AWS may send both notifications at the same time.

How to create such nodes and the limitations of their use are described in "How to create preemptible nodes in AWS?".

Replacing the OS image

The OS image of the nodes is set by the `ami` parameter of the AWSInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the AWSClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from "How do I replace the OS image in a cloud cluster?" .

Additional resources

- Module `cloud-provider-aws` documentation
- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in Amazon Web Services

This section covers the following additional aspects of integrating Deckhouse Platform (DP) with AWS:

- Connecting cloud disks using CSI.
- Automatic StorageClass creation.
- Using LoadBalancer.
- Accessing via bastion host.
- Connecting manually created CloudStatic nodes.

Storage (CSI and StorageClass)

DP integrates with AWS storage via CSI. This allows the cluster to automatically provision and attach disks to its nodes.

StorageClasses are created automatically for the following disk types:

- `gp3`
- `gp2`
- `sc1`
- `st1`

Disk types `io1` and `io2` are also supported with IOPS and throughput settings in ModuleConfig.

To exclude unnecessary StorageClasses from the cluster, specify filters in `settings.storageClass.exclude`:

```
settings:
  storageClass:
    exclude:
    - sc.*
    - st1
```

You can also define and configure custom StorageClasses explicitly, including parameters such as `iops`, `throughput`, and `type`:

```
settings:
  storageClass:
    provision:
    - name: fast-io
      type: io2
      iopsPerGB: "50"
```

Expanding volume size

To resize a volume (for example, when running low on disk space), follow these steps:

1. Modify the `spec.resources.requests.storage` parameter in the corresponding PersistentVolumeClaim object. The operation is performed automatically and usually takes less than a minute. You can monitor progress with the following command:

   ```
   kubectl describe pvc <claim-name>
   ```

2. After resizing, wait at least 6 hours and ensure the volume status is `in-use` or `available`. Only then you can safely perform additional size changes. See AWS documentation for details.

Load balancing

DP supports LoadBalancer Services via AWS Load Balancer Controller.

To control how AWS resources are created, use annotations on the Service object:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-12345,subnet-67890"
```

Where:

- `service.beta.kubernetes.io/aws-load-balancer-type`: If set to none, only a target group is created (no

LoadBalancer).
- `service.beta.kubernetes.io/aws-load-balancer-backend-protocol`: Only applicable when `aws-load-balancer-type` is `none`. Specifies the protocol for communication with the target group. Supported values:
  - `tcp` (default)
  - `tls`
  - `http`
  - `https`

> **i** When you change this annotation, `cloud-controller-manager` will attempt to recreate the target group. If it's already associated with an NLB or ALB, the deletion will fail, and the controller will enter a retry loop. To avoid this, manually detach the load balancer from the group.
>
> If Ingress nodes are not available in all zones, explicitly specify subnets using `aws-load-balancer-subnets`.

Configuring LoadBalancer when Ingress nodes are not in all zones

If your AWS cluster does not have Ingress nodes in every zone, you must explicitly list the subnets to be used by the Load Balancer via the `service.beta.kubernetes.io/aws-load-balancer-subnets` annotation:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-foo,subnet-bar"
```

This is especially important when manually configuring Ingress controllers or using non-standard node layout.

To retrieve the current list of subnets used in your DP setup, run the following:

```
kubectl -n d8-system exec svc/deckhouse-leader -c deckhouse \
  -- deckhouse-controller module values cloud-provider-aws -o json | \
  jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

Connecting CloudStatic nodes

To connect manually created EC2 instances to a DP cluster, follow these steps:

1. Attach the IAM role `<prefix>-node`.
2. Attach the security group `<prefix>-node`.
3. Add the following tags:

   ```
   "kubernetes.io/cluster/<cluster_uuid>" = "shared"
   "kubernetes.io/cluster/<prefix>" = "shared"
   ```

To get the `cluster_uuid`, use the following command:

```
kubectl -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
```

To get the `prefix`, use the following command:

```
d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
```

If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | \
  jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix
```

Additional resources

- Global custom resources
- Module `cloud-provider-aws` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with Amazon Web Services

Working with Amazon Elastic Container Registry (ECR)

To allow cluster nodes to access private Amazon ECR repositories, follow these steps:

1. Define read permissions for container images in the repository policies. It’s important that the `Principal` field includes the IAM Role attached to the Deckhouse Platform (DP) nodes.

   Example policy:

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "RepositoryRead",
         "Effect": "Allow",
         "Principal": {
           "AWS": "arn:aws:iam::xxx:role/xxx-node"
         },
         "Action": [
           "ecr:BatchCheckLayerAvailability",
           "ecr:BatchGetImage",
           "ecr:DescribeImages",
           "ecr:DescribeRepositories",
           "ecr:GetAuthorizationToken",
           "ecr:GetDownloadUrlForLayer",
           "ecr:ListImages",
           "ecr:ListTagsForResource"
         ]
       }
     ]
   }
   ```

2. Set this policy via the AWS Console. Go to **Amazon ECR → Private registry → Repositories →** select the desired repository **→ Permissions**.

3. Add `ecr:GetAuthorizationToken` to `additionalRolePolicies` in your AWSClusterConfiguration so that nodes can automatically retrieve an authorization token to pull images:

   ```
   additionalRolePolicies:
     - ecr:GetAuthorizationToken
     - ecr:BatchGetImage
     - ecr:DescribeRepositories
   ```

The additionalRolePolicies parameter lets you extend the set of IAM actions granted to EC2 instances managed by DP. This is particularly useful if access to the following is required:

- Amazon ECR
- Amazon S3
- Other AWS services that require specific permissions

If this parameter is not set, IAM roles will only include the basic actions:

```
ec2:DescribeTags
ec2:DescribeInstances
```

Additional resources

- Module `cloud-provider-aws` provider configuration

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Microsoft Azure

Requirements

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To ensure Deckhouse Platform (DP) works correctly with Microsoft Azure, the following conditions must be met:

- The selected region must support Availability Zones.
- All virtual machines must have the `cloud-init` package installed. After the VMs start, the following services must be active:
  - `cloud-config.service`
  - `cloud-final.service`

- `cloud-init.service`

⚠ Starting with version 1.77, new clusters in Azure use the Cilium CNI by default with pod traffic tunneling over VXLAN. In existing clusters, the CNI stays unchanged.

Make sure all nodes run Linux kernel 5.8 or newer and that firewall rules allow UDP traffic between nodes. For the list of ports, see Network interaction of the platform components; for CNI settings, see Internal network configuration.

Accessing the Azure API

To manage Azure resources from DP, you need a service account with the `Contributor` role within the appropriate subscription. Follow these steps:

1. Install Azure CLI and log in:

   ```
   export SUBSCRIPTION_ID=$(az login | jq -r '.[0].id')
   ```

2. Create a service account:

   ```
   az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --name
   "DeckhouseCANDI"
   ```

   Example output:

   ```
   {
    "appId": "<CLIENT_ID>",
    "displayName": "DeckhouseCANDI",
    "password": "<CLIENT_SECRET>",
    "tenant": "<TENANT_ID>"
   }
   ```

   Specify the output values in the AzureClusterConfiguration object:

Field	Value from the output
`clientId`	`appId`
`clientSecret`	`password`
`tenantId`	`tenant`

3. Authenticate in Azure CLI using the created service account's credentials:

   ```
   az login --service-principal -u <CLIENT_ID> -p <CLIENT_SECRET> --tenant <TENANT_ID>
   ```

ℹ By default, the `clientSecret` is valid for 1 year, and no automatic renewal is supported. To set a longer expiration period, refer to the official Azure documentation.

Additional resources

- Module `cloud-provider-azure` provider configuration

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in Microsoft Azure

This section describes the cluster deployment layout in Microsoft Azure infrastructure and the associated parameters.

Standard

Standard is the supported deployment layout with the following characteristics:

- A separate resource group is created for the cluster.
- Each instance is assigned a public IP address by default (used only for accessing the internet).
- Up to 64,000 SNAT ports are available per public IP address.
- A NAT Gateway is supported and billable. It allows using static public IP addresses for SNAT.
- Public IPs can be assigned to both master nodes and nodes created via Terraform.
- If the master node does not have a public IP, a bastion host and VNet peering between clusters are required.
- Peering between the cluster's VNet and other VNets is supported.

When a cluster is created, Deckhouse Platform (DP) creates a Network Security Group (NSG) named after the cluster prefix and associates it with the node subnet.

The following rules will be created:

- `AllowIcmp`: Allows incoming traffic over the ICMP protocol from any source.
- `AllowSsh`: Allows incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList`. If the list is not set, traffic is allowed from any source.

You cannot attach a pre-created custom NSG to nodes through the DP parameters. To restrict SSH access, use `sshAllowList`.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"      # Required.
vNetCIDR: 10.50.0.0/16              # Required.
subnetCIDR: 10.50.0.0/24            # Required.
standard:
  natGatewayPublicIpCount: 1         # Optional (0 by default).
masterNodeGroup:
  replicas: 1
  zones: ["1"]                     # Optional (["1"] by default).
  instanceClass:
    machineSize: Standard_F4         # Required.
    diskSizeGb: 32
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140  # Required.
    enableExternalIP: false          # Optional (true by default).
provider:
  subscriptionId: "<SUBSCRIPTION_ID>"  # Required.
  clientId: "<CLIENT_ID>"            # Required.
  clientSecret: "<CLIENT_SECRET>"     # Required.
  tenantId: "<TENANT_ID>"            # Required.
  location: "westeurope"             # Required.
peeredVNets:                        # Optional.
  - resourceGroupName: kube-bastion   # Required.
    vnetName: kube-bastion-vnet       # Required.
```

Replacing the OS image

The OS image of the nodes is set by the `urn` parameter of the AzureInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the AzureClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Additional resources

- Module `cloud-provider-azure` provider configuration
- Module `cloud-provider-azure` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in Microsoft Azure

Storage

When running on Azure, Deckhouse Platform (DP) automatically creates the following StorageClass resources:

Name	Disk type
`managed-standard`	Standard_LRS
`managed-standard-ssd`	StandardSSD_LRS
`managed-premium`	Premium_LRS

You can additionally:

- Disable unneeded storage classes using the `exclude` parameter.
- Define custom StorageClass resources with the target throughput and IOPS settings.

Example configuration in a ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
      - name: managed-ultra-ssd
        type: UltraSSD_LRS
        diskIOPSReadWrite: 600
        diskMBpsReadWrite: 150
      exclude:
      - managed-standard.*
      - managed-premium
```

Load balancing

DP automatically creates LoadBalancer resources in Azure when using Kubernetes Service objects of the LoadBalancer type.

Additional features:

- NAT Gateway is supported. You can explicitly define the number of public IP addresses for SNAT using the `natGatewayPublicIpCount` parameter.
- VNet peering can be configured, including with a bastion host or other VNets in the cloud.
- Service Endpoints are supported. They provide secure and direct connections to Azure services without using public IPs. For more details, refer to Integration with Microsoft Azure services.

Additional resources

- Module `cloud-provider-azure` parameters
- Module `cloud-provider-azure` provider configuration

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with Microsoft Azure services

Deckhouse Platform (DP) leverages Azure cloud features for full integration with Kubernetes. When running a cluster on Azure, it automatically:

- Creates network routes for the PodNetwork.
- Configures external LoadBalancers for Kubernetes Service objects.
- Removes nodes from the cluster that no longer exist in the cloud.
- Updates node metadata to reflect the current configuration.
- Provisions disks for nodes via CSI.
- Connects the required CNI network (`cni-cilium` is used).
- Makes AzureInstanceClass definitions available for virtual machines, which can be used later in NodeGroup configurations.

> All outgoing traffic from the cluster comes through LoadBalancers. If no LoadBalancer is configured to handle UDP traffic, all outgoing UDP traffic will be blocked, which can affect NTP utilities such as `ntpdate`, `chrony`, etc. To resolve this, manually add a rule for any UDP port to an existing LoadBalancer or create a Kubernetes Service of the LoadBalancer type with a UDP port.

Support for Service Endpoints

DP supports connections to Azure services via Service Endpoints. These endpoints:

- Allow access to Azure services without using public IP addresses.
- Route traffic over optimized Azure backbone infrastructure.
- Simplify access control and improve security.

List of supported Service Endpoints:

```
Microsoft.AzureActiveDirectory
Microsoft.AzureCosmosDB
Microsoft.ContainerRegistry
Microsoft.CognitiveServices
Microsoft.EventHub
Microsoft.KeyVault
Microsoft.ServiceBus
Microsoft.Sql
Microsoft.Storage
Microsoft.Storage.Global
Microsoft.Web
```

Specify the required services using the `serviceEndpoints` parameter in the AzureClusterConfiguration object.

Additional resources

- Module `cloud-provider-azure` custom resources
- Module `cloud-provider-azure` provider configuration
- Module `cni-cilium` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Google Cloud Platform

To manage Google Cloud resources using Deckhouse Platform, you need to create a Service Account.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> ⚠ Starting with version 1.77, new clusters in Google Cloud use the Cilium CNI by default with pod traffic tunneling over VXLAN. In existing clusters, the CNI stays unchanged.
>
> Make sure all nodes run Linux kernel 5.8 or newer and that firewall rules allow UDP traffic between nodes. For the list of ports, see Network interaction of the platform components; for CNI settings, see Internal network configuration.

Creating a service account

You can find detailed instructions on how to create a service account in the official Google Cloud documentation.

> ⚠ A created `service account key` cannot be recovered. If lost, it must be deleted and recreated.

Setup via Google Cloud console

Go to the Google Cloud console, select your project, and create a new service account (you can also choose an existing one).

The created service account must be assigned the following roles:

```
Compute Admin
Service Account User
Network Management Admin
```

You can assign roles during service account creation or modify them later.

To generate the `service account key` in JSON format, on the service accounts page, click the three vertical dots in the **Actions** column and select **Manage keys**. Then click **Add key → Create new key → Key type → JSON**.

Setup via Google Cloud CLI

Install and initialize the Google Cloud CLI by following the official instructions.

To create a service account using the CLI, follow these steps:

1. Export environment variables:

   ```
   export PROJECT_ID=sandbox
   export SERVICE_ACCOUNT_NAME=deckhouse
   ```

2. Set the project:

```
gcloud config set project $PROJECT_ID
```

3. Create a service account:

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

4. Assign roles to the created service account:

```
for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
   --role=${role}; done
```

   A list of required roles:

```
roles/compute.admin
roles/iam.serviceAccountUser
roles/networkmanagement.admin
```

5. Verify the assigned roles:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
     --filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Generate a `service account key`:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
     ~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

Using the service account

The generated `service account key` must be specified in the `provider.serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"` section of the GCPClusterConfiguration resource.

Example:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

Additional resources

- Module `cloud-provider-gcp` provider configuration

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in Google Cloud Platform

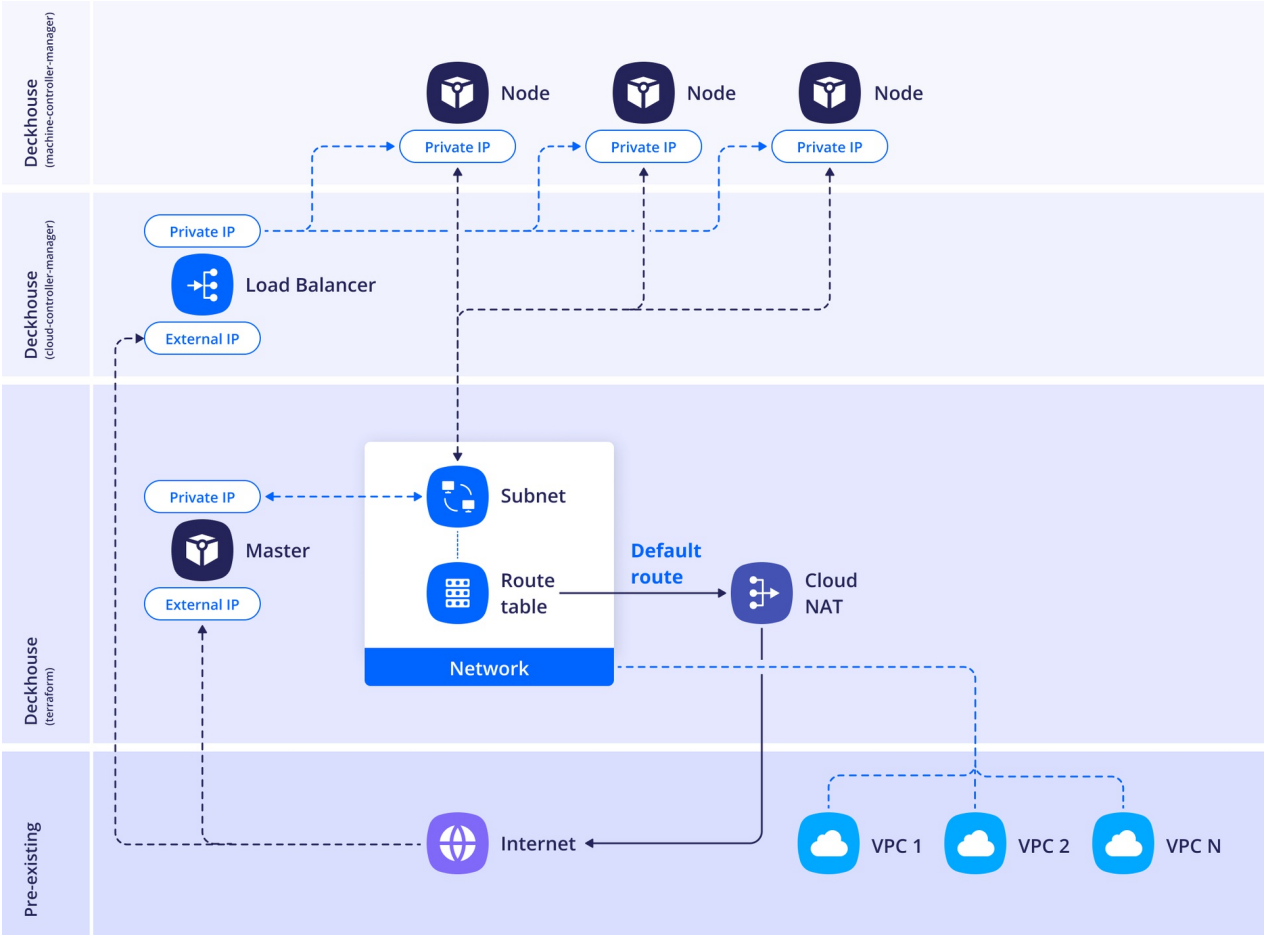
Layouts

This section describes the possible node placement layouts in Google Cloud Platform (GCP) infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Deckhouse Platform (DP) supports two layouts for deploying resources in GCP.

Standard

- A dedicated VPC is created for the cluster along with Cloud NAT.
- Cluster nodes do not have public IP addresses by default.
- Public IP addresses can be assigned to static and master nodes:
  - In this case, One-to-One NAT is used to map the public IP address to the node's internal IP address (Cloud NAT will not be used in this scenario).
- If the master node does not have a public IP address, an additional instance with a public IP address (for example, a bastion host) is required for cluster installation and access.
- Peering can be configured between the cluster's VPC and other VPCs.



When a cluster is created, DP creates the following firewall rules in the cluster VPC:

- `<CLUSTER_PREFIX>-ssh-and-ping`: Allows incoming traffic over the ICMP and TCP (port `22`) protocols to nodes with the `<CLUSTER_PREFIX>` network tag from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- `<CLUSTER_PREFIX>-intercommunication`: Allows any traffic between nodes with the `<CLUSTER_PREFIX>` network tag, as well as from the pod subnet (`podSubnetCIDR`).

Custom firewall rules are applied to nodes through additional network tags (`additionalNetworkTags`):

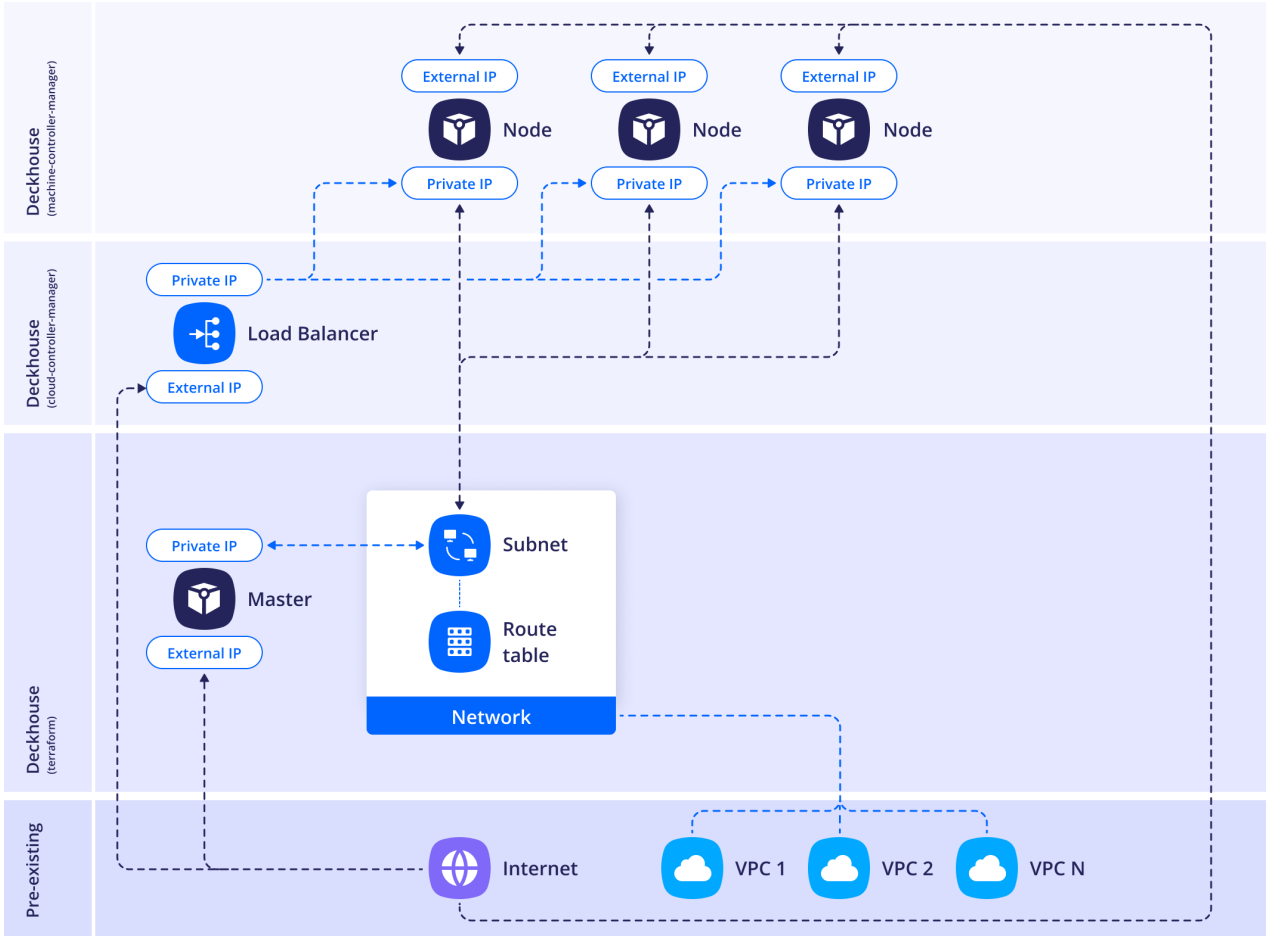
- For master nodes, in the `masterNodeGroup.instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalNetworkTags` parameter of the GCPInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Optional. Listed addresses will be used for
  # addressing Cloud NAT.
  cloudNATAddresses:
  - example-address-1
  - example-address-2
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional. A list of GCP VPC Networks used by Kubernetes VPC
# Network to connect over peering.
peeredVPCs:
- default
sshKey: "<SSH_PUBLIC_KEY>"  # Required.
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                      # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    disableExternalIP: false       # Optional. The master node has externalIP by default.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                      # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    disableExternalIP: true        # Optional. The nodes don't have externalIP by default.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: static
provider:
  region: europe-west4           # Required.
  serviceAccountJSON: |          # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

WithoutNAT

- A dedicated VPC is created for the cluster. All cluster nodes are assigned public IP addresses.
- Peering can be configured between the cluster’s VPC and other VPCs.



When a cluster is created, DP creates the following firewall rules in the cluster VPC:

- `<CLUSTER PREFIX>-ssh-and-ping`: Allows incoming traffic over the ICMP and TCP (port `22`) protocols to nodes with the `<CLUSTER PREFIX>` network tag from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- `<CLUSTER_PREFIX>-intercommunication`: Allows any traffic between nodes with the `<CLUSTER_PREFIX>` network tag, as well as from the pod subnet (`podSubnetCIDR`).

Custom firewall rules are applied to nodes through additional network tags (`additionalNetworkTags`):

- For master nodes, in the `masterNodeGroup.instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalNetworkTags` parameter of the GCPInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional. A list of GCP VPC Networks used by Kubernetes VPC
# Network to connect over peering.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: static
provider:
  region: europe-west4            # Required.
  serviceAccountJSON: |           # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

Configuration

Integration with GCP is handled via the GCPClusterConfiguration resource, which describes the cloud cluster configuration in GCP and is used by the cloud provider when the control plane is hosted in the cloud. The responsible DP module configures itself automatically based on the selected layout.

To update the configuration of a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After modifying node-related parameters, you must run the `dhctl converge` command for the changes to take effect.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

Machine provisioning and parameters are configured in the NodeGroup custom resource, where the instance class for the node group is specified (the `cloudInstances.classReference` parameter). For GCP, the instance class is a GCPInstanceClass custom resource that defines the machine parameters.

DP also automatically creates StorageClasses that cover all available disk types in GCP:

Disk type	Replication	StorageClass name
standard	none	`pd-standard-not-replicated`
standard	regional	`pd-standard-replicated`
balanced	none	`pd-balanced-not-replicated`
balanced	regional	`pd-balanced-replicated`
ssd	none	`pd-ssd-not-replicated`
ssd	regional	`pd-ssd-replicated`

You can exclude unnecessary StorageClasses by specifying them in the `exclude` parameter.

Enabling nested virtualization

To run virtual machine workloads (e.g., KVM-based VMs) inside GCP instances, enable nested virtualization.

> **i** Only specific machine types support nested virtualization. See the GCP documentation for the list of compatible types.

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  nestedVirtualization: true
```

Adding additional disks

To attach additional disks to instances (for example, for LINSTOR, Ceph, NFS storage nodes, and similar solutions), specify them in the `additionalDisks` parameter:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
  - sizeGb: 200
    type: pd-ssd
  - sizeGb: 500
    type: pd-standard
    autoDelete: true
```

Adding CloudStatic nodes to the cluster

To add manually created virtual machines to the cluster as nodes, add a `Network Tag` matching the cluster prefix.

You can find the cluster prefix by running the following command:

```
d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
```

If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
  | base64 -d | grep prefix
```

Preemptible nodes

Preemptible virtual machines in Google Cloud are called preemptible instances. To make a node group order them, set the `preemptible` parameter of the GCPInstanceClass resource to `true`.

How to create such nodes and how they behave is described in “How to create preemptible nodes in GCP?”.

Replacing the OS image

The OS image of the nodes is set by the `image` parameter of the GCPInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the GCPClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?” .

Additional resources

- Global custom resources
- Module `cloud-provider-gcp` documentation
- Module `control-plane-manager` parameters
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Private cloud providers

Deckhouse Platform provides built-in tools for integration with private clouds deployed in your own infrastructure. These tools let you automatically provision virtual machines and connect them to a cluster, deploy clusters within private cloud environments, and scale resources as needed.

- Basis Dynamix
- Built-in virtualization
- Huawei Cloud
- OpenStack
- VK Private Cloud

The following sections describe how you can configure integration with each of these private cloud platforms.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in VK Cloud

To manage resources in VK Private Cloud using Deckhouse Platform, you must connect to the VK Private Cloud API.

The list of VK Private Cloud API services required for deployment is available in the Configuration section.

The user credentials required to access the VK Private Cloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of VK Private Cloud and using it are available in the VK Private Cloud documentation.

If you’re using a cloud provider’s implementation of the VK Private Cloud API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in VK Cloud

List of required OpenStack services

The following VK Private Cloud services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to order a Load Balancer.

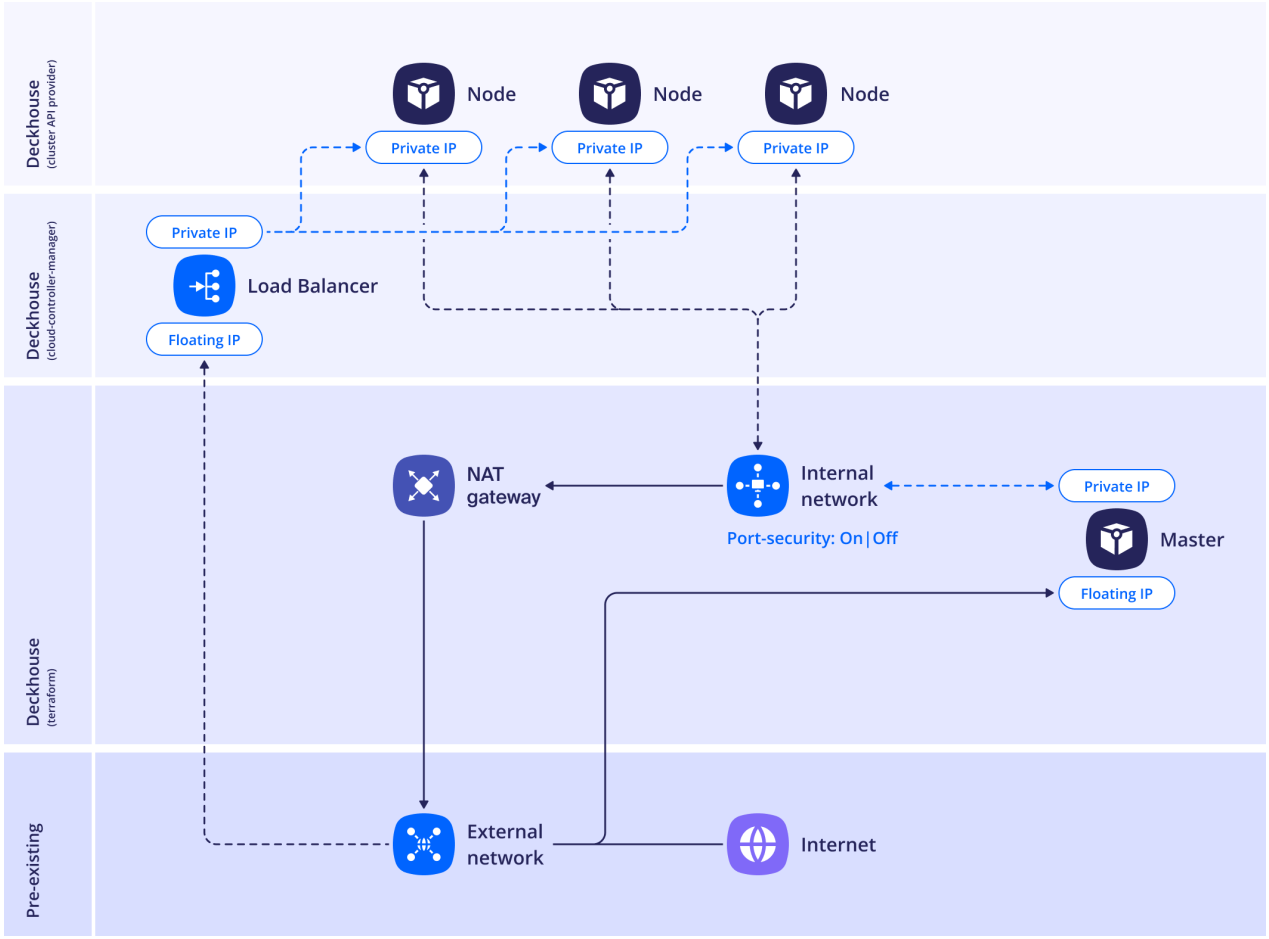
API endpoints and ports are listed in the official documentation.

Layouts

This section describes the possible node placement layouts in VK Private Cloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



> ⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared               # Required.
  bastion:
    zone: ru2-b                           # Optional.
    volumeType: fast-ru-2b                  # Optional.
    instanceClass:
      flavorName: m1.large                  # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
```

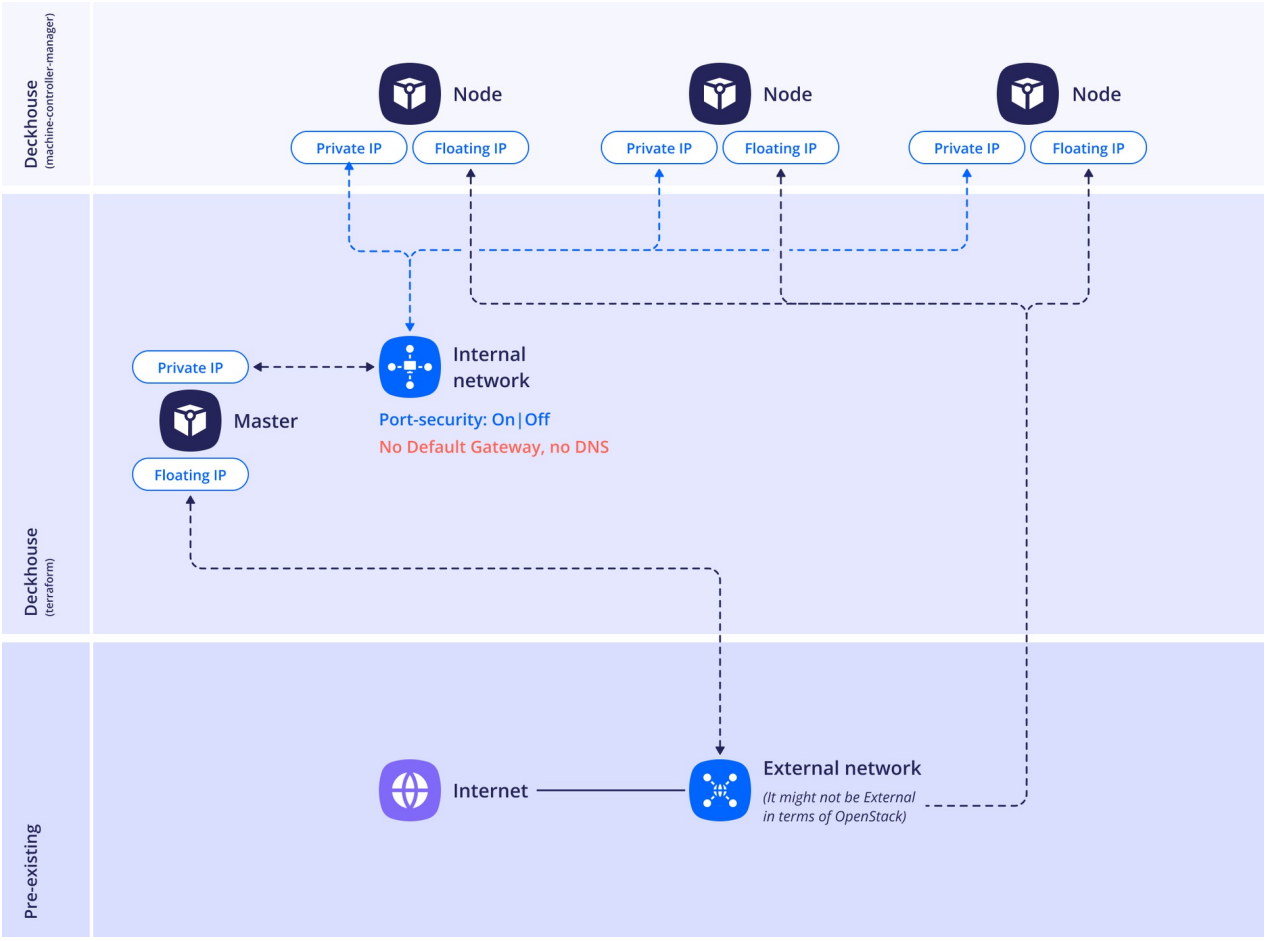
```
     rootDiskSize: 50                    # Optional, 50 GB by default.
     additionalTags:
       severity: critical                # Optional.
       environment: production            # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

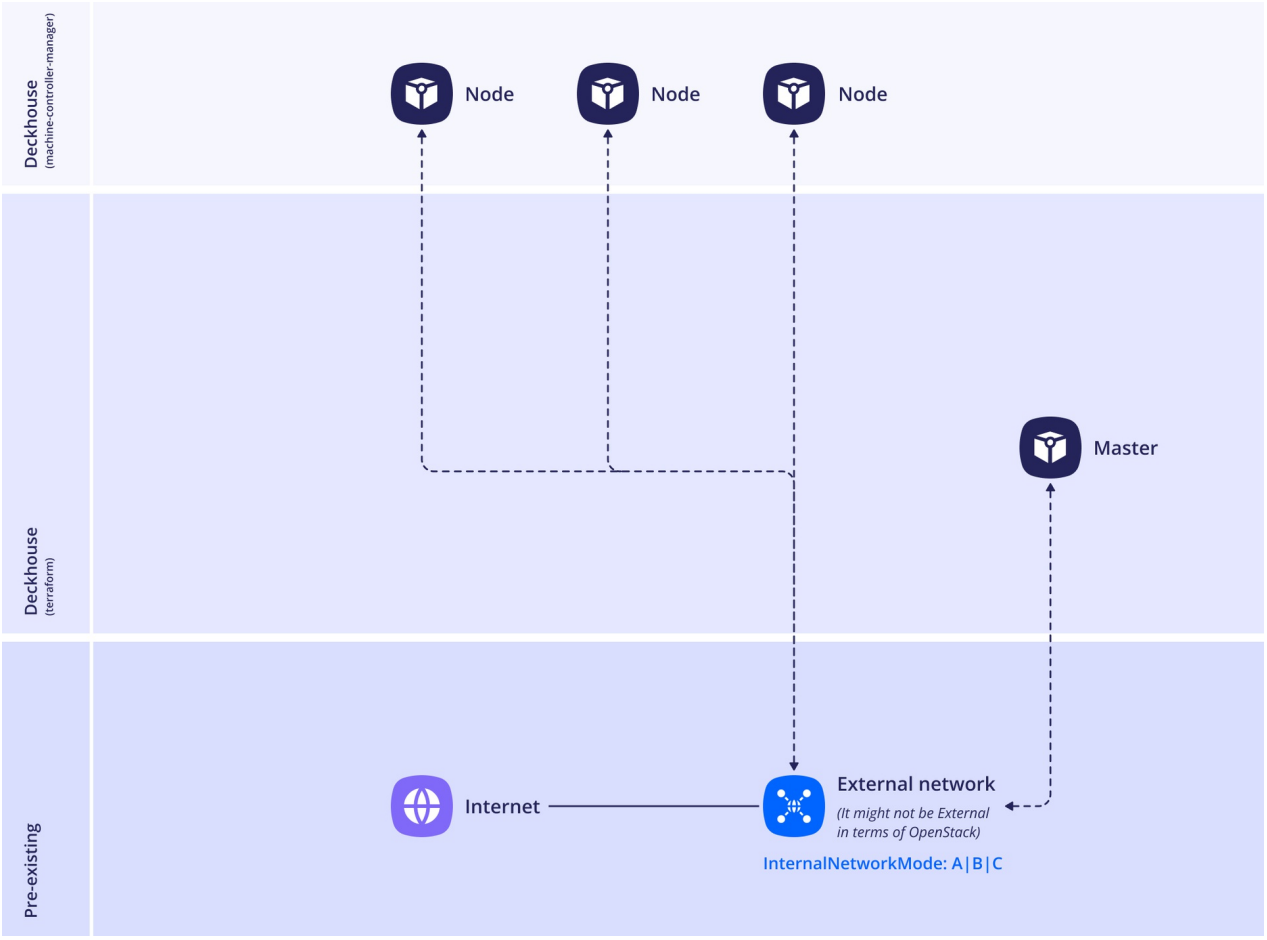
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

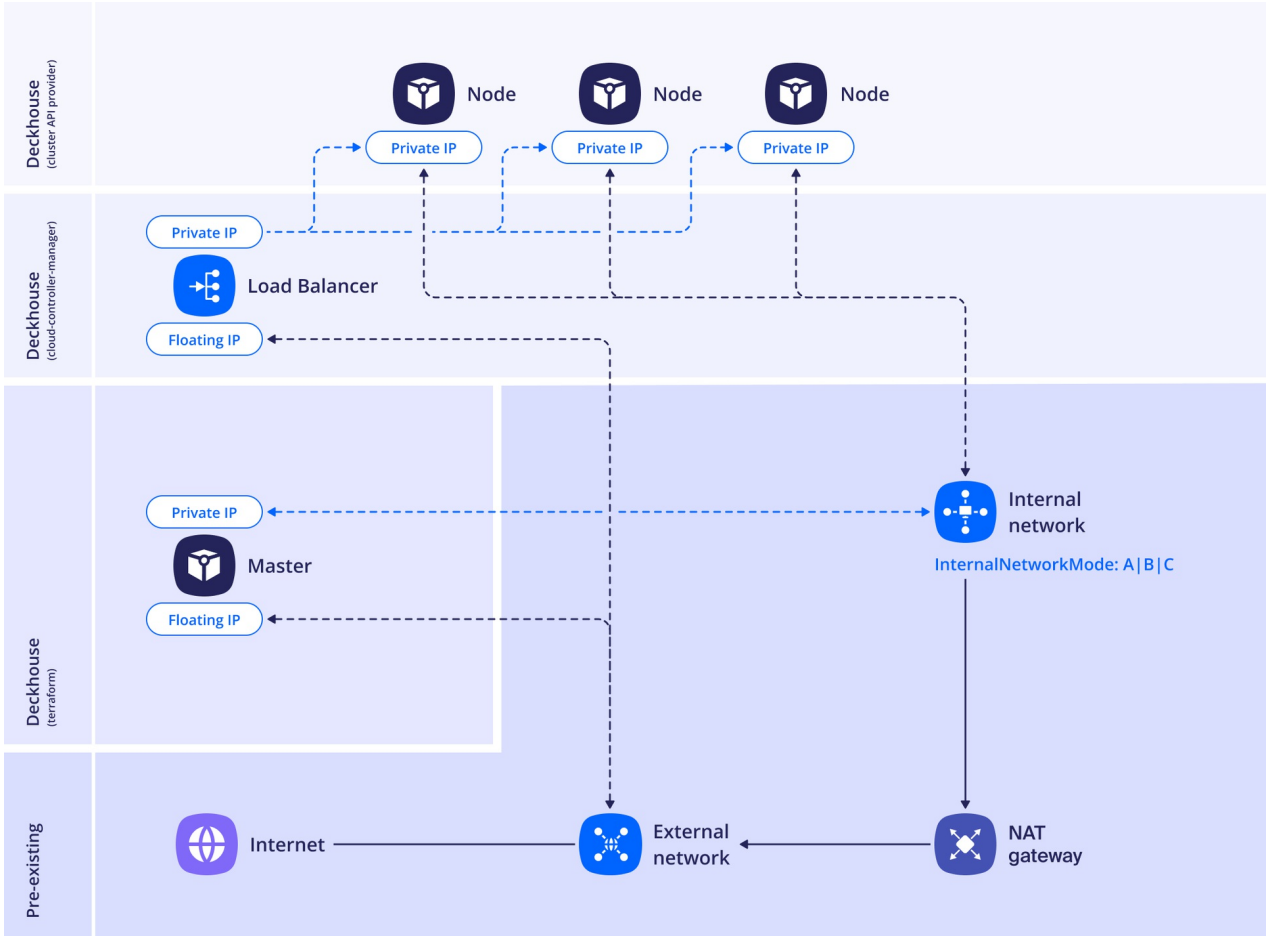
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with VK Private Cloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in VK Private Cloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VK Private Cloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to VK Private Cloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the VK Private Cloud API.

   > **i** The method of obtaining the openrc file may vary depending on the VK Private Cloud provider. If your provider offers the standard interface for VK Private Cloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if VK Private Cloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in VK Private Cloud

A node’s disk can be either local or network-attached. In VK Private Cloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In VK Private Cloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:

- If specified in the OS image, it will be used.
- If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new VK Private Cloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in OpenStack

To manage resources in OpenStack using Deckhouse Platform, you must connect to the OpenStack API.

The list of OpenStack API services required for deployment is available in the Configuration section.

The user credentials required to access the OpenStack API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OpenStack and using it are available in the OpenStack documentation.

If you're using a cloud provider's implementation of the OpenStack API, the process for obtaining the openrc file may differ.

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in OpenStack

List of required OpenStack services

The following OpenStack services must be available for Deckhouse Platform (DP) to operate:

Service	API Version
Identity (Keystone)	v3

Service	API Version
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

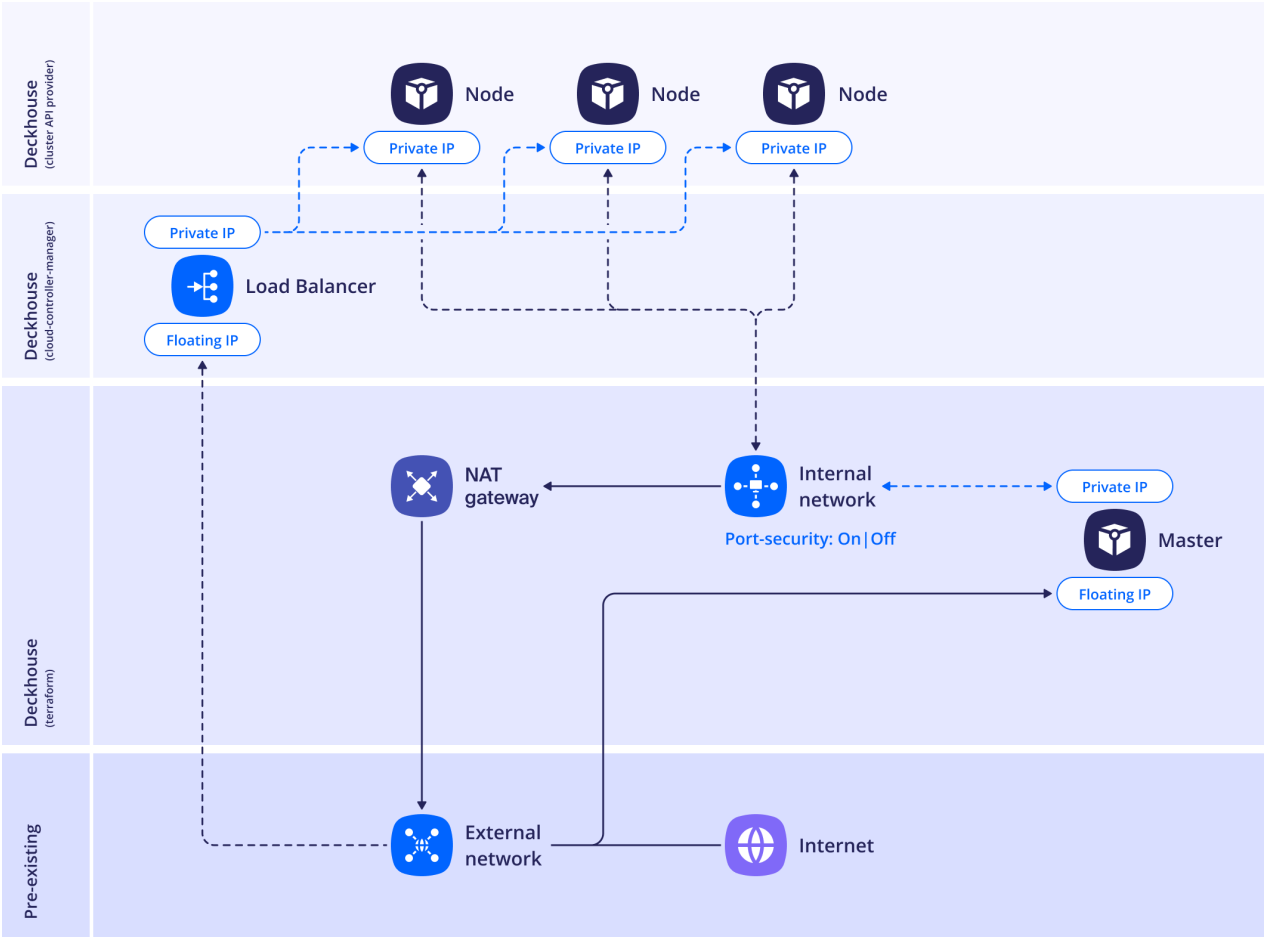
* If you need to order a Load Balancer.

Layouts

This section describes the possible node placement layouts in OpenStack infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.



⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.

- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
```

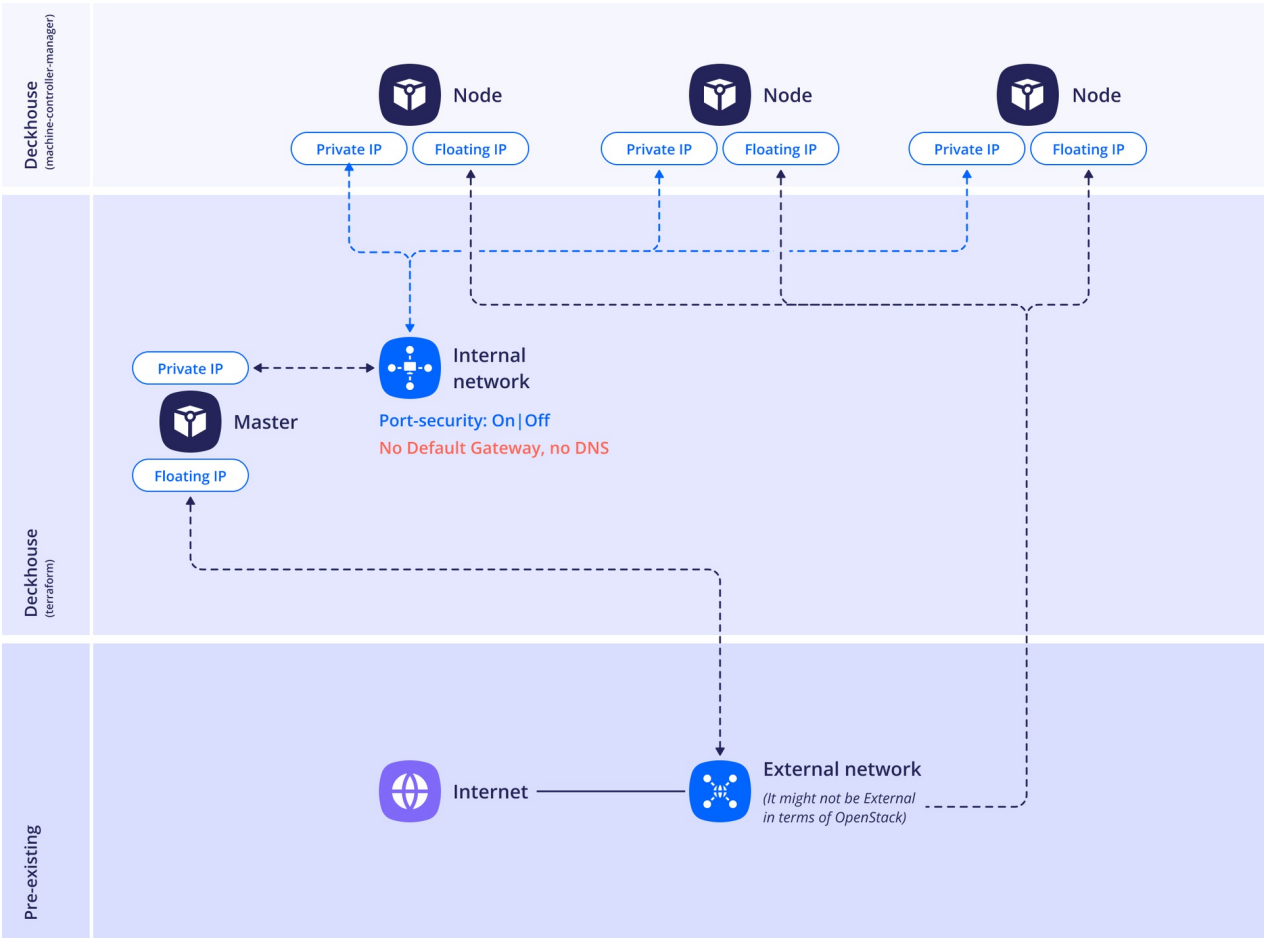
```
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from the CIDRs listed in `sshAllowList` (default `0.0.0.0/0`).
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.
- Allow any incoming traffic from nodes in the same security group.

In addition to the group created by DP, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the

OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

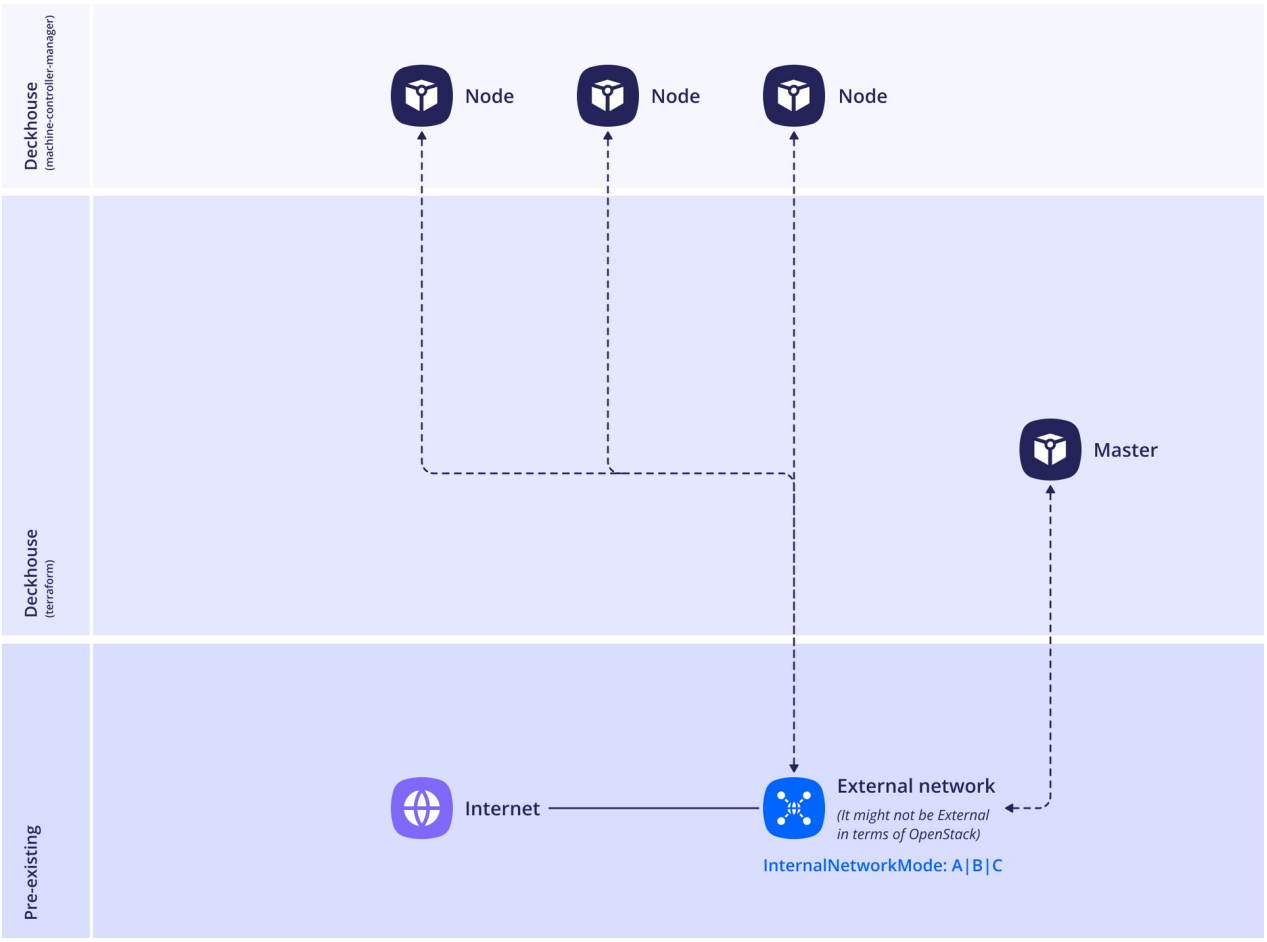
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

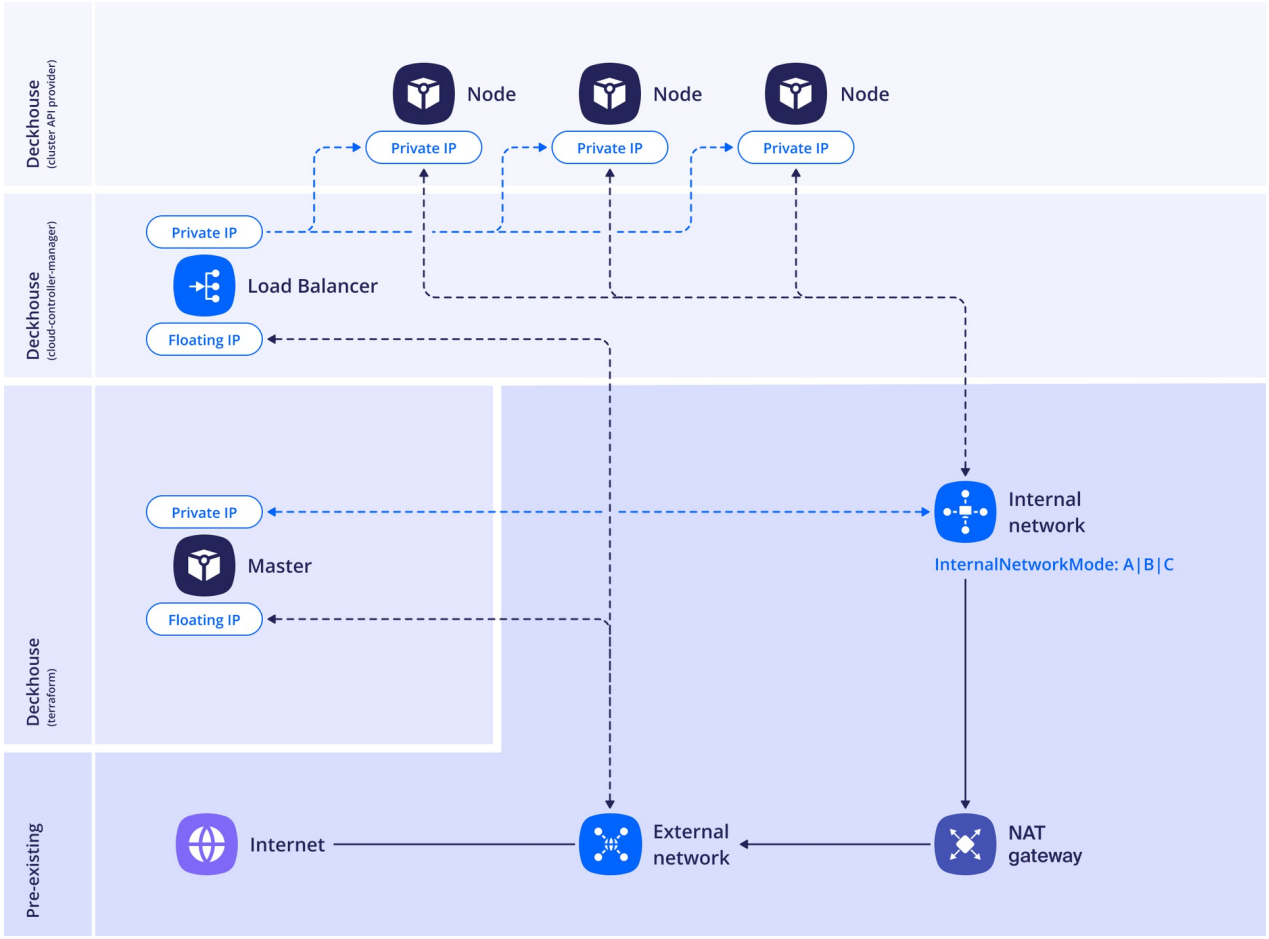
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.



In this layout, DP does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OpenStack is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OpenStack and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

```
d8 system edit provider-cluster-configuration
```

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OpenStack cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

> **⚠** When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Uploading an image to OpenStack

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OpenStack API.

   > **i** The method of obtaining the openrc file may vary depending on the OpenStack provider. If your provider offers the standard interface for OpenStack, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OpenStack does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the OpenStackInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the OpenStackClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Cinder CSI resilience during re-authentication in OpenStack

The Cinder CSI driver supports OpenStack re-authentication with service catalog refresh. This improves the reliability of

volume operations in `cinder-csi-plugin` pods that have been running for a long time without being restarted.

When a token expires or a `401 Unauthorized` response is received, the driver re-authenticates with OpenStack and updates the data used to access OpenStack API services. As a result, PersistentVolume operations, including volume attachment and detachment, continue without requiring CSI driver pods to be restarted.

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without being restarted, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OpenStack

A node’s disk can be either local or network-attached. In OpenStack terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OpenStack, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:

  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider's default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OpenStack image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Huawei Cloud

Requirements

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To ensure proper operation of Deckhouse Platform (DP) with Huawei Cloud, make sure of the following:

- The `cloud-init` package is installed on the virtual machines.
- After the VMs start, the following services must be active:
  - `cloud-config.service`
  - `cloud-final.service`
  - `cloud-init.service`

Accessing the Huawei Cloud API

DP uses the Huawei Cloud API to manage resources. To configure access, you need to create an IAM user and assign the necessary permissions.

Creating a user group

To create a user group and assign policies, follow these steps:

1. Go to the **Identity and Access Management (IAM)** section.
2. Open the **User Groups** tab and click **Create User Group**.
3. Specify a group name (for example, `deckhouse`) and click **OK**.
4. Select the created group and go to the **Permissions** tab.
5. Click **Authorize** and assign the following policies:
   - `ECS Admin`
   - `VPC Administrator`
   - `NAT Admin`
   - `ELB FullAccess`
   - `DEW KeypairFullAccess`
6. Confirm your selections by clicking **Next**, then **OK**, and finalize with **Finish**.

Creating an IAM user

To create an IAM user, follow these steps:

1. Go to the **Users** tab and click **Create User**.
2. Enter a username (for example, `deckhouse`).
3. Under **Access type**, select **Programmatic access**, and ensure that **Management console access** is disabled.
4. Under **Credential Type**, choose **Access key**.
5. Click **Next**, select the previously created group, and then click **Create**.
6. Download the **Access Key ID** and **Secret Access Key**. These credentials are required to access the Huawei Cloud API and cannot be recovered later.

> **i** Make sure the saved keys are securely stored, as they are required to access the cloud API.

JSON policies

Below are the policy contents in JSON format:

ECS Admin policy

```
{
"Version": "1.1",
"Statement": [
{
    "Action": [
    "ecs:*:*",
    "evs:*:get",
    "evs:*:list",
    "evs:volumes:create",
    "evs:volumes:delete",
    "evs:volumes:attach",
    "evs:volumes:detach",
    "evs:volumes:manage",
    "evs:volumes:update",
    "evs:volumes:use",
    "evs:volumes:uploadImage",
    "evs:snapshots:create",
    "vpc:*:get",
    "vpc:*:list",
    "vpc:networks:create",
    "vpc:networks:update",
    "vpc:subnets:update",
    "vpc:subnets:create",
    "vpc:ports:*",
    "vpc:routers:get",
    "vpc:routers:update",
    "vpc:securityGroups:*",
    "vpc:securityGroupRules:*",
    "vpc:floatingIps:*",
    "vpc:publicIps:*",
    "ims:images:create",
    "ims:images:delete",
    "ims:images:get",
    "ims:images:list",
    "ims:images:update",
    "ims:images:upload"
    ],
    "Effect": "Allow"
}
]
}
```

VPC Administrator policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "vpc:vpcs:*",
                "vpc:routers:*",
                "vpc:networks:*",
                "vpc:subnets:*",
                "vpc:ports:*",
                "vpc:privateIps:*",
                "vpc:peerings:*",
                "vpc:routes:*",
                "vpc:lbaas:*",
                "vpc:vpns:*",
                "ecs:*:get",
                "ecs:*:list",
                "elb:*:get",
                "elb:*:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

NAT Admin policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "nat:*:*",
                "vpc:*:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

DEW KeypairFullAccess policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kps:domainKeypairs:*",
                "ecs:serverKeypairs:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ELB FullAccess policy

```
{
  "Version": "1.1",
  "Statement": [
      {
          "Action": [
              "elb:*:*",
              "vpc:*:get*",
              "vpc:*:list*",
              "ecs:*:get*",
              "ecs:*:list*"
          ],
          "Effect": "Allow"
      }
  ]
}
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in Huawei Cloud

This section describes the cluster layouts in Huawei Cloud infrastructure and the associated parameters.

List of required Huawei Cloud services

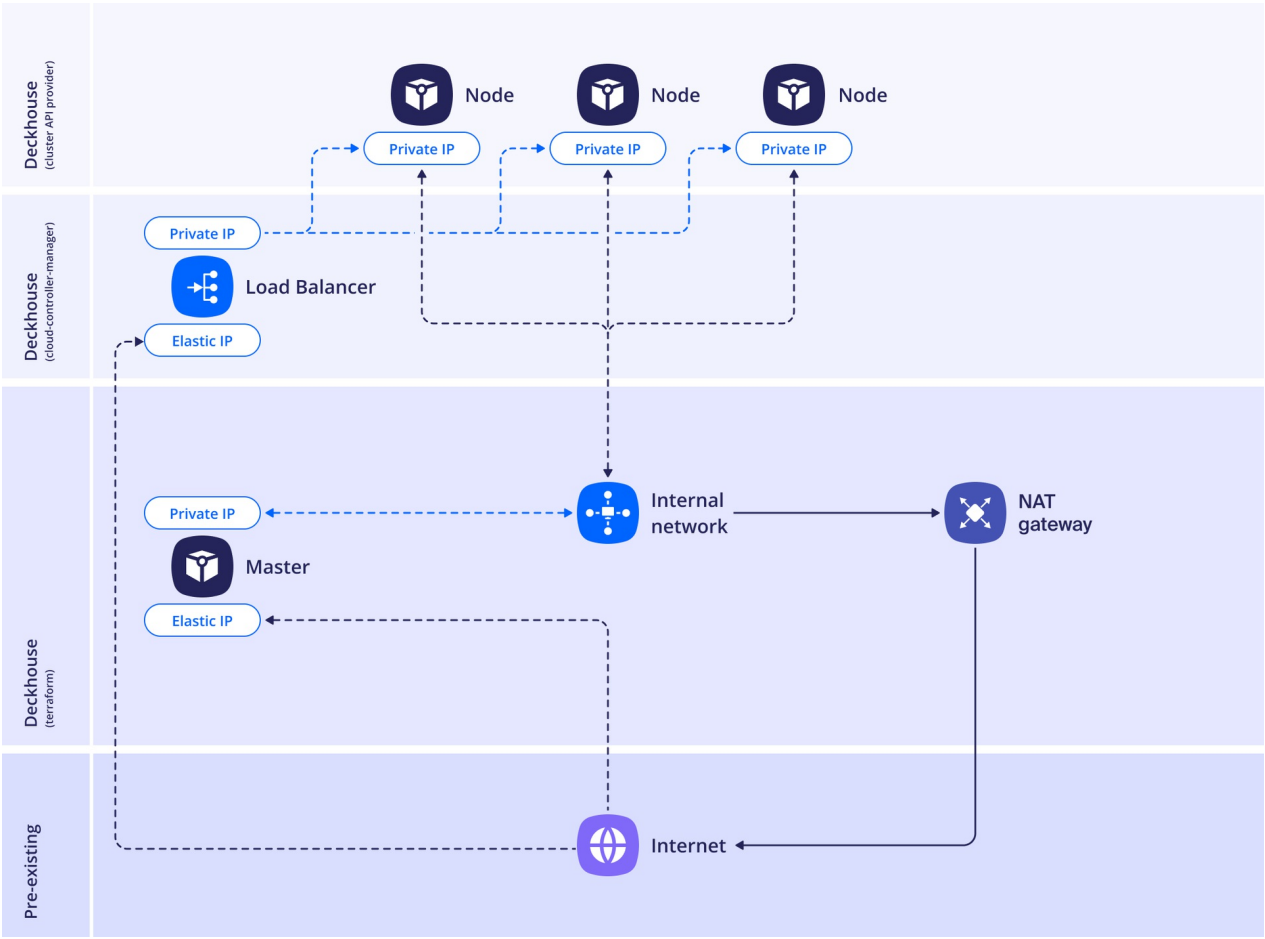
A list of services required for Deckhouse Platform (DP) to work in Huawei Cloud:

Service	API version
Identity	v3
Compute	v2.1
Network (VPC)	v1/v2
Network (OpenStack Neutron API)	v2.0
Block Storage	v3
Load Balancing (ELB) *	v3

* Required if load balancers need to be provisioned in the cluster.

Standard

- An internal cluster network is created with a gateway to the public network.
- An Elastic IP can be assigned to the master node.
- Nodes managed by the Cluster API do not have public IP addresses.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from `0.0.0.0/0`.
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.

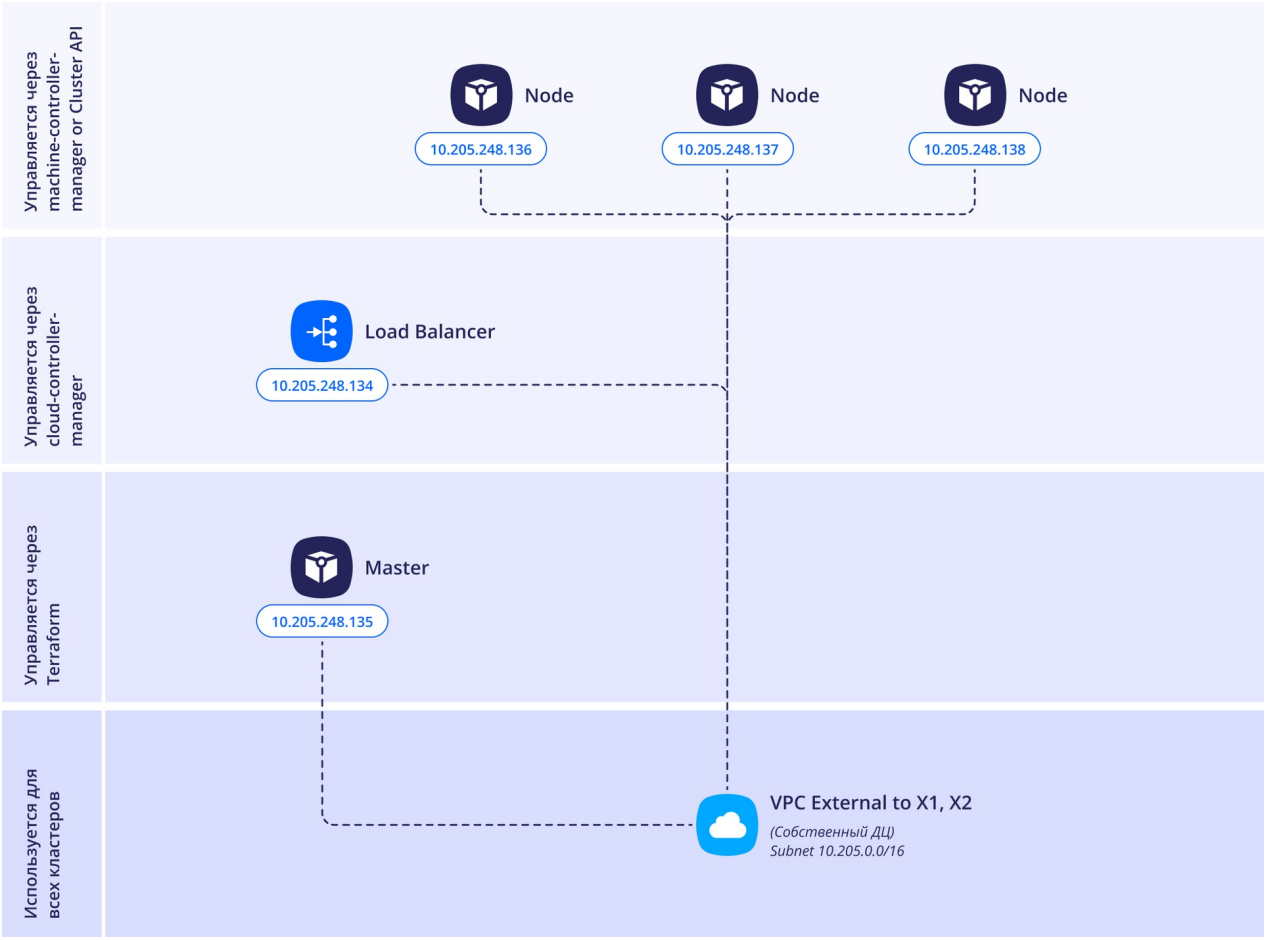
Attach custom security groups for CloudEphemeral nodes in the HuaweiCloudInstanceClass resource via `spec.securityGroups`. They are applied together with the group created by DP.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering

- An existing VPC network and subnet are used.
- Virtual machines are connected to the specified subnet.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. DP creates the group named after the cluster prefix and assigns it to the nodes.

The following inbound rules will be created:

- Allow incoming traffic over the TCP protocol on port `22` from `0.0.0.0/0`.
- Allow incoming traffic over the ICMP protocol from `0.0.0.0/0`.
- Allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default.

Attach custom security groups for CloudEphemeral nodes in the HuaweiCloudInstanceClass resource via `spec.securityGroups`. They are applied together with the group created by DP.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4
```

Additional resources

- Module `cloud-provider-huaweicloud` provider configuration
- Module `cloud-provider-huaweicloud` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in Huawei Cloud

Storage

Deckhouse Platform provisions disks in Huawei Cloud using the CSI driver. Storage type is configured via parameters in the HuaweiCloudClusterConfiguration resource, specifically the `volumeTypeMap` field.

Example configuration:

```
masterNodeGroup:
  volumeTypeMap:
    ru-moscow-1a: SSD
```

When the `rootDiskSize` parameter is specified, the same disk type is used for the boot (root) volume.

The following parameters can also be set in the HuaweiCloudInstanceClass:

- `rootDiskSize`: Disk size in GiB.
- `rootDiskType`: Type of the root disk (SSD, GPSSD, SAS, etc.).

Example:

```
spec:
  rootDiskSize: 50
  rootDiskType: GPSSD
```

It is recommended that you use the fastest available disk types supported in the cloud provider’s region.

Load balancing

Load balancing in Huawei Cloud is handled via Elastic Load Balancer (ELB) services. To access the load balancer API, the IAM policy must include the `ELB FullAccess` permission.

In the Standard layout, the `enableEIP` parameter is available to assign Elastic IPs to master nodes.

By default, nodes managed by the Cluster API do not receive public IP addresses.

Additional resources

- Module `cloud-provider-huaweicloud` provider configuration
- Module `cloud-provider-huaweicloud` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with Huawei Cloud services

Deckhouse Platform leverages Huawei Cloud’s capabilities to operate Kubernetes clusters. The following features are

supported:

- Resource management in Huawei Cloud via `cloud-controller-manager`.
- Disk provisioning using the CSI driver.
- Integration with the `node-manager` module, allowing the use of HuaweiCloudInstanceClass in NodeGroup.

Working with InstanceClass

The HuaweiCloudInstanceClass resource is used to define the parameters of virtual machines. It is referenced by NodeGroup.

Example resource:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

Replacing the OS image

The OS image of the nodes is set by the `imageName` parameter of the HuaweiCloudInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the HuaweiCloudClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new image, follow the steps from “How do I replace the OS image in a cloud cluster?”.

Additional resources

- Module `cloud-provider-huaweicloud` custom resources
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in Basis Dynamix

> The Basis Dynamix integration is experimental. Compatibility with future versions isn’t guaranteed and the behavior may change.

The integration provides the following capabilities:

- Ordering and removing virtual machines via the Basis Dynamix API
- Using prepared cloud images to deploy nodes
- Configuring virtual machine parameters (CPU, RAM, disk, network)
- Support for external and internal networks, including CIDR and DNS settings
- Support for multiple layouts with an external or a combined network
- Disk placement by a storage policy

Requirements

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To integrate Deckhouse Platform with Basis Dynamix, you need the following:

- Basis Dynamix 4.6 or newer, otherwise the preflight check stops the installation
- Access to the API controller and to the Basis Dynamix SSO
- The account name and the application parameters (`appId` and `appSecret`)
- Account access to the Basis Dynamix user API groups
- An OS cloud image uploaded to the cloud
- An external network (`externalNetwork`) and, if required, the internal network parameters (CIDR, DNS)
- The name of a storage policy available to the account (`storagePolicy`)
- A public SSH key for accessing the cluster nodes

Preparing a cloud image

To deploy virtual machines, Deckhouse Platform uses OS cloud images prepared by vendors for virtual environments. Such images usually include:

- `cloud-init`

- Virtual hardware drivers
- Guest agents

Use official cloud images distributed in the `.img`, `.qcow2`, and similar formats.

To add an image to Basis Dynamix:

1. Go to “Images” → “Template images”.
2. Upload the prepared cloud image.

Configuring access to the portal

If your infrastructure has no DNS server, add the IP addresses and domains of the Basis Dynamix portal to the `cloud-init` template manually, so that the system works correctly when the nodes start.

1. Edit the `hosts` file template located in the following directory:

   ```
   /etc/cloud/templates/
   ```

2. Add the IP address and domain name entries specific to your operating system and configuration. This may include entries for the Basis Dynamix portal, such as:

   ```
   <IP address> <domain name>
   ```

3. After the changes, clean up the `cloud-init` state:

   ```
   cloud-init clean
   ```

4. Create a template from the prepared virtual machine. This is required for the `/etc/hosts` file to be generated correctly on the first boot.

Additional resources

- Module `cloud-provider-dynamix` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

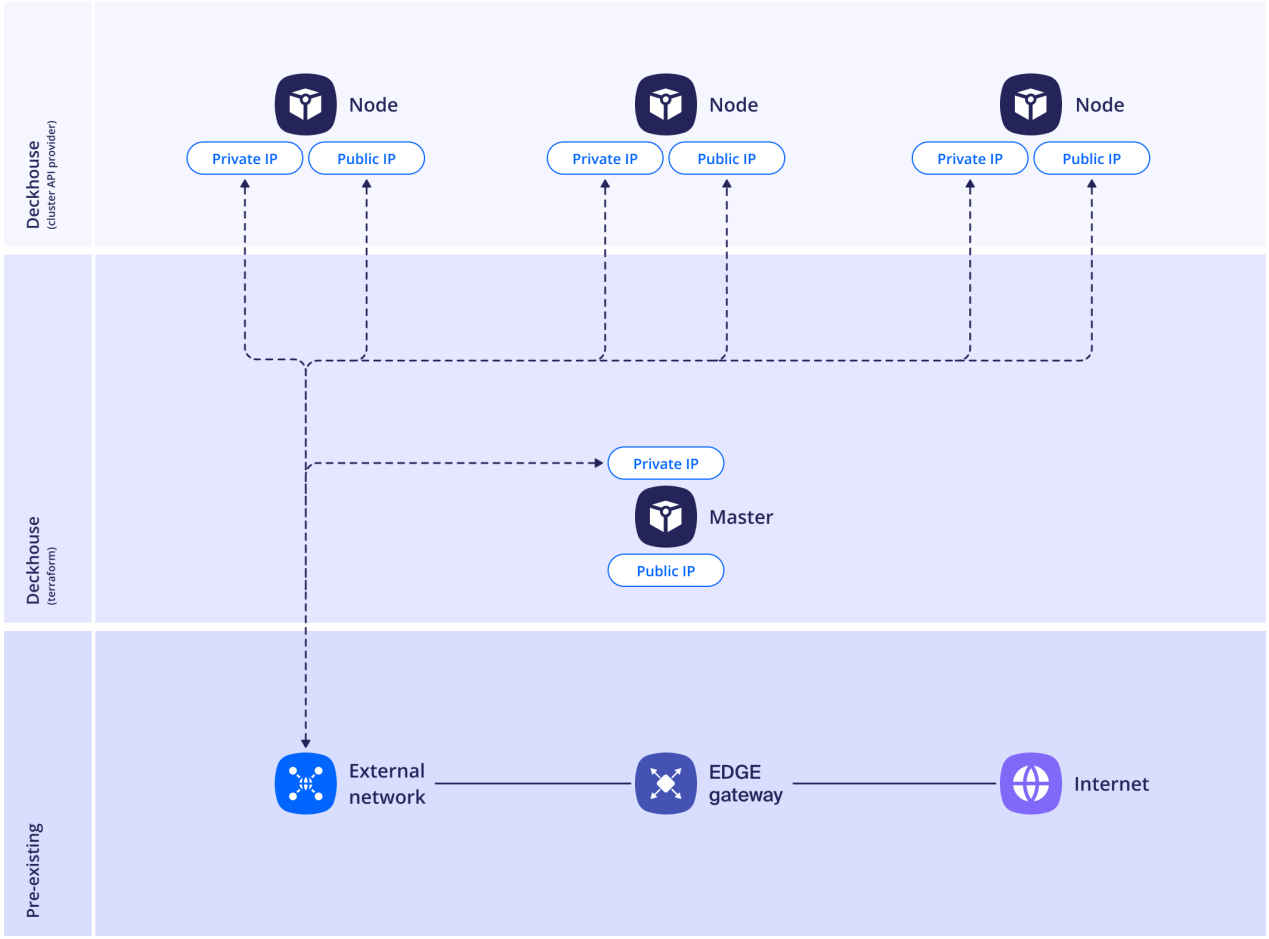
Layouts and configuration in Basis Dynamix

Deckhouse Platform supports two layouts in the Basis Dynamix cloud:

- Standard: A layout that uses only an external network
- StandardWithInternalNetwork: A layout with an internal (private) network and DNS servers

Both layouts let you manage cluster node placement, network settings, the OS image, and data storage.

Standard



If the external network doesn’t provide DNS server addresses to the nodes, set them in the `nameservers` parameter.

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
storagePolicy: storage_policy01
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

StandardWithInternalNetwork

Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
storagePolicy: storage_policy01
nodeNetworkCIDR: "10.241.32.0/24"
nameservers:
  - "10.0.0.10"
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

Required parameters

- `layout`: Layout
- `sshPublicKey`: Public key for accessing the nodes
- `location`: Name of the Basis Dynamix location where the cluster resource group is created
- `account`: Account name in the cloud
- `storagePolicy`: Name of the storage policy used by default for virtual machine disks
- `provider.controllerUrl`, `oAuth2Url`, `appId`, `appSecret`: API access parameters
- `imageName`: OS image name
- `externalNetwork`: External network name
- `nodeNetworkCIDR`: Address space of the internal network (only for the `StandardWithInternalNetwork` layout)

⚠ Changing `storagePolicy` recreates nodes of the `CloudEphemeral` type. The nodes that are recreated are listed in the Changing the storage policy section.

To apply the parameter changes, run `dhctl converge`.

Additional resources

- Module `cloud-provider-dynamix` provider configuration

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in Basis Dynamix

Storage

Virtual machine disk placement in Basis Dynamix is defined by a storage policy. It sets the storage endpoint and pool pairs available to the account, as well as an IOPS limit. Basis Dynamix chooses the exact placement of a disk within the policy.

The `storagePolicy` parameter at the root of DynamixClusterConfiguration sets the storage policy for the disks of all virtual machines in the cluster. To use a different storage policy for a node group, set `storagePolicy` in its `instanceClass`, including `masterNodeGroup.instanceClass`, or in a DynamixInstanceClass. The `rootDiskSizeGb` parameter sets the root disk size of each virtual machine in gigabytes.

Example configuration:

```
storagePolicy: storage_policy01
masterNodeGroup:
  replicas: 1
  instanceClass:
    rootDiskSizeGb: 50
```

The module creates a StorageClass for every `ENABLED` storage policy. The StorageClass naming rules and the way to exclude StorageClasses from the cluster are described in the module FAQ.

Disable a storage policy in Basis Dynamix only when the cluster nodes and volumes no longer use it. When a policy is disabled, the module detaches it from the cluster resource group and deletes its StorageClass.

Changing the storage policy

The storage policy of a disk is set when the virtual machine is created and doesn't change for an existing machine. Therefore, when the storage policy changes, the module recreates nodes of the `CloudEphemeral` type:

- When `storagePolicy` changes in a DynamixInstanceClass: The nodes of the groups that use this DynamixInstanceClass.
- When `storagePolicy` changes at the root of DynamixClusterConfiguration: The nodes of all groups, including groups with their own storage policy.

The module doesn't recreate master nodes and the nodes of the `nodeGroups` groups. A new storage policy applies to such a node only when the node is created again.

Load balancing

The Basis Dynamix platform doesn't provide a built-in load balancer. To handle inbound traffic to a Deckhouse Platform cluster, the following approaches are recommended:

1. An external load balancer. If your infrastructure has an external load balancer (hardware or software), configure it to forward ports 80 and 443 to the cluster's frontend nodes.
2. Using MetalLB. You can use MetalLB in L2 mode to provide fault-tolerant load balancing.

Recommendations:

- Allocate a separate L2 network with DHCP and internet access.
- Configure the IP address range MetalLB will announce addresses from.
- Make sure this network is connected to the cluster's frontend nodes.
- Leave the network interfaces empty in the VirtualMachine Template configuration. Deckhouse Platform creates them automatically.

> **i** Support for BGP mode depends on the network infrastructure and isn't guaranteed on Basis Dynamix.

Configuring a Service of the LoadBalancer type

To configure a Service of the LoadBalancer type, add the following annotations to the Service manifest:

```
metadata:
  annotations:
    dynamix.cpi.flant.com/internal-network-name: <internal_name>
    dynamix.cpi.flant.com/external-network-name: <external_name>
```

Both annotations are required:

- `dynamix.cpi.flant.com/internal-network-name`: The name of the internal network in Basis Dynamix
- `dynamix.cpi.flant.com/external-network-name`: The name of the external network in Basis Dynamix

The terms "internal network" and "external network" are used in the context of Basis Dynamix. The external network doesn't have to be public and may use private IP addresses.

If one of the annotations isn't specified, `cloud-controller-manager` fails to process the Service with an error.

Additional resources

- Module `cloud-provider-dynamix` documentation
- Module `node-manager` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with Basis Dynamix cloud services

Deckhouse Platform integrates with the Basis Dynamix cloud platform and uses DynamixInstanceClass resources to describe the characteristics of virtual machines deployed in the cluster.

Key capabilities

- Ordering and removing virtual machines via the Basis Dynamix API
- Configuring virtual machine parameters, including the number of CPUs, memory size, and root disk size
- Specifying the OS template (image name) and the storage policy used for disk placement
- Connecting to external networks
- Using multiple node groups with individual parameters

Example DynamixInstanceClass:

```
apiVersion: deckhouse.io/v1
kind: DynamixInstanceClass
metadata:
  name: frontend
spec:
  numCPUs: 4
  memory: 8192
  rootDiskSizeGb: 40
  imageName: alt-p10-cloud-x86_64.img
  storagePolicy: storage_policy01
  externalNetwork: extnet_vlan_1700
```

It is referenced by the NodeGroup's `cloudInstances.classReference` parameter.

Recommendations

- Place OS images in the "Images" → "Template images" section of the Basis Dynamix portal.
- Use image names that exactly match the `imageName` values in DynamixInstanceClass.
- Make sure the selected storage policy is available to the account and has the `ENABLED` status.
- Verify that virtual machines have access to the internet and to DNS servers.

Cloud integration provides automatic scaling, configuration, and node management according to the parameters set in DynamixInstanceClass and the cluster configuration.

Additional resources

- Module `cloud-provider-dynamix` custom resources
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtualization systems

Deckhouse Platform (DP) supports the following virtualization systems:

- Deckhouse Virtualization Platform (DVP);
- VMware Cloud Director;
- VMware vSphere;
- zVirt.

The following sections describe how you can configure integration with each of these virtualization systems.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with built-in virtualization

Deckhouse Platform integrates with the built-in virtualization infrastructure and uses DVPInstanceClass resources to define the characteristics of virtual machines created as part of the cluster.

Key features:

- Management of built-in virtualization resources via the `cloud-controller-manager module`
- Provisioning of disks using the CSI storage component
- Integration with the `node-manager` module to support DVPInstanceClass when defining a NodeGroup

> The virtualization integration is enabled automatically for all cloud clusters deployed in built-in virtualization. No additional configuration is required.

Load balancers

Virtualization supports Service objects of the `LoadBalancer` type. For every such object, `cloud-controller-manager` creates a `LoadBalancer` Service in the parent cluster and directs traffic to the nodes of the child one. The object is created in the namespace set by the `namespace` parameter, next to the cluster virtual machines.

The external address is assigned by the parent cluster, so a balancer controller must run there, for example the `metallb` module. The controller serves only those Service objects that have a resolved balancer class.

To set the default class, create a MetalLoadBalancerClass resource in the parent cluster with the `isDefault` parameter set to `true`. Such a class serves the Service objects with no class specified, so you can leave the class out in the child cluster.

To pick another class, set it in the `spec.loadBalancerClass` parameter of the Service object in the child cluster. The platform copies the value to the parent cluster along with the `externalIPs` parameter and the deprecated `loadBalancerIP` parameter.

⚠ The `spec.loadBalancerClass` parameter is immutable. Set the class when creating the Service object, otherwise the object has to be recreated.

With no balancer class, the Service object stays in the `Pending` state, and `cloud-controller-manager` reports no errors. Check the state of the object in the parent cluster:

```
d8 k -n <NAMESPACE> get svc
```

Replace `<NAMESPACE>` with the namespace set by the `namespace` parameter.

Load balancer network policy

If the `networkPolicy` parameter is set to `Isolated`, the parent cluster restricts ingress traffic to the cluster virtual machines (VMs). This value is used by default. To let the balancer traffic reach the nodes, `cloud-controller-manager` creates a NetworkPolicy resource for every balancer in the same namespace of the parent cluster.

The policy allows ingress traffic from any source to the NodePort ports of the Service object in the child cluster over the protocols specified in this object. The policy applies only to the VMs the balancer directs traffic to, so every balancer has its own policy, even if several balancers use the same nodes.

`cloud-controller-manager` overwrites manual changes to the policy the next time it updates the balancer. The policy is deleted together with the balancer Service object in the parent cluster.

To find the balancer policy, follow these steps:

1. Get the UID of the Service object in the child cluster:

   ```
   d8 k -n <SERVICE_NAMESPACE> get svc <SERVICE_NAME> -o jsonpath='{.metadata.uid}'
   ```

   Replace `<SERVICE_NAMESPACE>` with the namespace of the Service object and `<SERVICE_NAME>` with its name.

2. Display the policy in the parent cluster:

   ```
   d8 k -n <NAMESPACE> get networkpolicy lb-<SERVICE_UID>
   ```

   Replace `<NAMESPACE>` with the namespace set by the `namespace` parameter and `<SERVICE_UID>` with the UID obtained in the previous step.

Additional resources

- Module `cloud-provider-dvp` custom resources
- Module `cloud-provider-dvp` parameters
- Module `metallb` documentation
- Module `node-manager` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in built-in virtualization

To interact with built-in virtualization resources, Deckhouse Platform components use the virtualization API. To configure access, create a user (ServiceAccount), assign the necessary permissions, and generate a kubeconfig.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

⚠ If the `update-hostname` module of the `cloud-init` package is not disabled, it is recommended to change its run frequency from `always` to `once-per-instance`.

To do this, modify the `update-hostname` module configuration in the `/etc/cloud/cloud.cfg` file:

```
cloud_init_modules:
  ...
  - [update-hostname, once-per-instance]
  ...
```

The `update-hostname` module can also be disabled completely by removing it from the `cloud_init_modules` module list in the `/etc/cloud/cloud.cfg` file.

Creating a user

Create a new user in the built-in virtualization cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-demo
  namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: sa-demo-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: sa-demo
type: kubernetes.io/service-account-token
EOF
```

Assigning a role

Assign a role to the created user in the built-in virtualization cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-demo-rb
  namespace: default
subjects:
  - kind: ServiceAccount
    name: sa-demo
    namespace: default
roleRef:
  kind: ClusterRole
  name: d8:namespace:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

If the virtualization cluster runs a DP version earlier than 1.78, the `d8:namespace:*` roles do not exist there. In this case, specify the `d8:use:role:manager` role in `roleRef.name`.

Generating a kubeconfig

Generate a kubeconfig to be used in the cluster initial configuration file:

```
cat <<EOF > kubeconfig
apiVersion: v1
clusters:
- cluster:
    server: https://<KUBE-APISERVER-URL>   # Replace this with the actual API server address for the
cluster.
  name: <CLUSTER-NAME>                     # Replace with the cluster name.
contexts:
- context:
    cluster: <CLUSTER-NAME>                # Replace with the cluster name.
    user: sa-demo
    namespace: default
  name: sa-demo-context
current-context: sa-demo-context
kind: Config
preferences: {}
users:
- name: sa-demo
  user:
    token: $(d8 k get secret sa-demo-token -n default -o json | jq -rc .data.token | base64 -d)
EOF
```

Encode the generated kubeconfig file using Base64 encoding (put it in the `d8-credentials` Secret in the `stringData.secret` field of the initial configuration file):

```
base64 kubeconfig | tr -d '\n'
```

Credentials Secret

The credentials for accessing the parent cluster API are stored in a separate Secret rather than in the ModuleConfig. The provider reads it when starting the components that access the DVP API.

The Secret is created during cluster installation together with the other resources of the initial configuration. If the module is added to a running cluster, the Secret is applied after the module creates the namespace, and the order is covered in the Hybrid cluster with DVP section. The Secret must meet the following requirements:

- The name is `d8-credentials` and the namespace is `d8-cloud-provider-dvp`.
- The type is `cloud-provider.deckhouse.io/credentials`.
- The `authScheme` field is set to `kubeconfig`.
- The `secret` field contains the Base64-encoded kubeconfig.

```
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-dvp
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: kubeconfig
  secret: <KUBE_CONFIG_BASE64>
```

Replace `<KUBE_CONFIG_BASE64>` with the Base64-encoded kubeconfig.

To change the credentials, update the `secret` field:

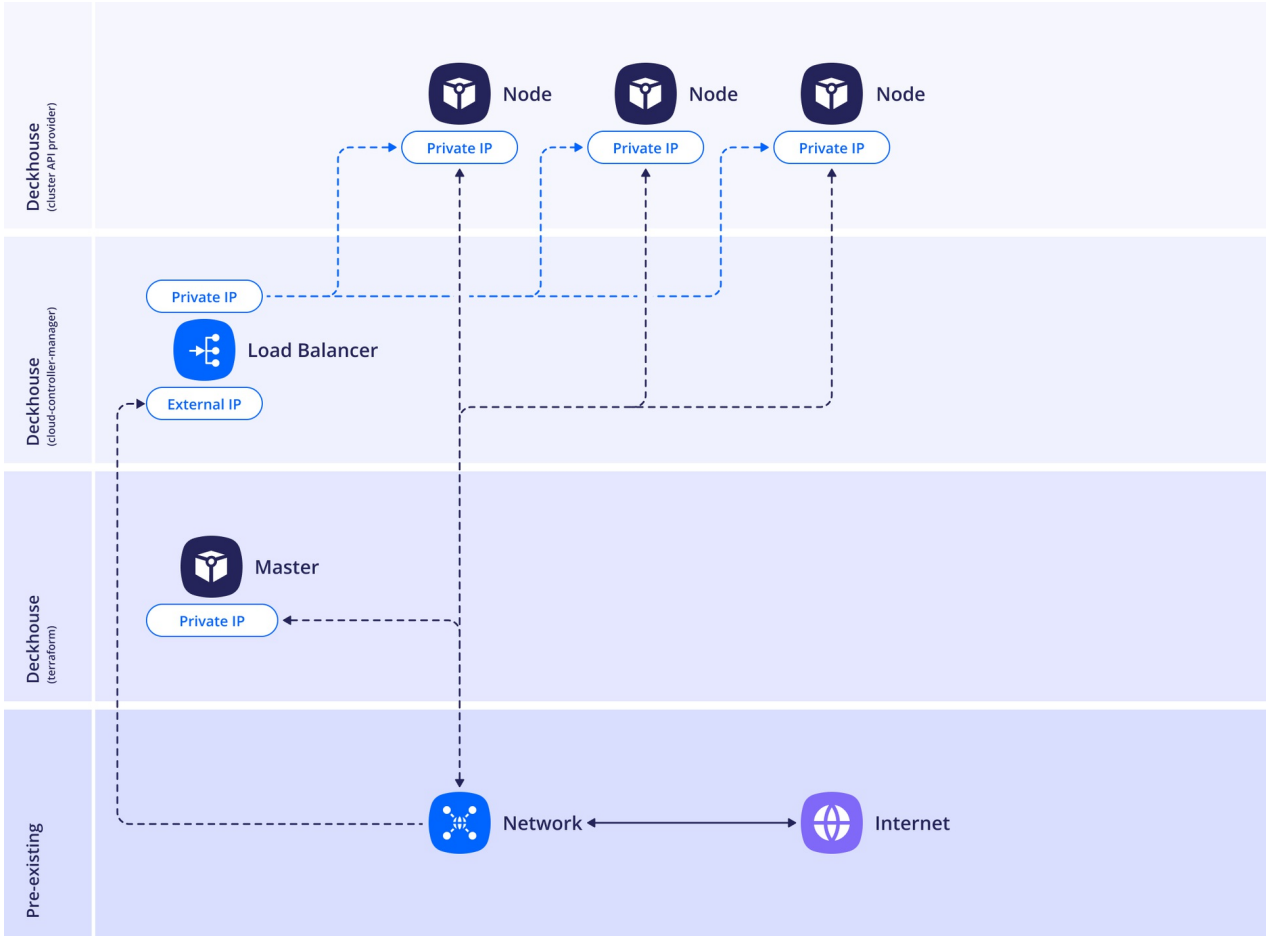
```
d8 k -n d8-cloud-provider-dvp edit secret d8-credentials
```

In clusters migrated to ModuleConfig, the kubeconfig is moved to this Secret from the `provider.kubeconfigDataBase64` parameter of the DVPClusterConfiguration resource.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in built-in virtualization

Standard



> ⚠ If the cluster was installed with the DVPClusterConfiguration schema, migration to ModuleConfig-based configuration is required. Until the migration is completed, the `D8CloudProviderDVPMigrationPending` alert may fire and Deckhouse updates may be blocked.
>
> See: How to migrate a cloud provider to ModuleConfig-based configuration.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  version: 2
  enabled: true
  settings:
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
        ipAddresses:
          master:
            - Auto
    provider:
      parameters:
        namespace: demo
---
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-dvp
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: kubeconfig
  secret: <KUBE_CONFIG_BASE64>
---
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: master
spec:
  virtualMachine:
    cpu:
      cores: 4
      coreFraction: 100%
    memory:
      size: 8Gi
    virtualMachineClassName: generic
  rootDisk:
    size: 50Gi
    storageClass: ceph-pool-r2-csi-rbd-immediate
    image:
      kind: ClusterVirtualImage
      name: ubuntu-2204
  etcdDisk:
    size: 15Gi
    storageClass: ceph-pool-r2-csi-rbd-immediate
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: master
    maxPerZone: 1
    minPerZone: 1
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
```

Configuration

The built-in virtualization cloud provider parameters are set in four resources:

- The ModuleConfig of the `cloud-provider-dvp` module defines the layout and the parent cluster connection

parameters.
- The `d8-credentials` Secret in the `d8-cloud-provider-dvp` namespace stores the credentials for accessing the parent cluster API. Its format is covered in the Credentials Secret section.
- The DVPInstanceClass resource describes the virtual machine parameters.
- The NodeGroup resource describes the node groups and references the instance class.

To modify the provider settings in a running cluster, edit the ModuleConfig:

```
d8 k edit mc cloud-provider-dvp
```

After changing node parameters, run the command:

```
dhctl converge
```

Example of a configuration with multiple parameters:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  version: 2
  enabled: true
  settings:
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: "<SSH_PUBLIC_KEY>"
        region: r1
        zones:
          - zone-a
          - zone-b
          - zone-c
        ipAddresses:
          master:
            - 10.66.30.100
            - 10.66.30.101
            - 10.66.30.102
    provider:
      parameters:
        namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-dvp
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: kubeconfig
  secret: ZXhhbXBsZQo=
---
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: master
spec:
  virtualMachine:
    cpu:
      cores: 1
      coreFraction: 100%
    memory:
      size: 4Gi
    virtualMachineClassName: generic
    additionalLabels:
      additional-vm-label: label-value
    additionalAnnotations:
      additional-vm-annotation: annotation-value
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system
    nodeSelector:
      nodeSelectorTerms:
```

```
      - matchExpressions:
          - key: beta.kubernetes.io/os
            operator: In
            values:
              - linux
  rootDisk:
    size: 10Gi
    storageClass: linstor-thin-r1
    image:
      kind: ClusterVirtualImage
      name: ubuntu-2204
  etcdDisk:
    size: 10Gi
    storageClass: linstor-thin-r1
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: master
    maxPerZone: 1
    minPerZone: 1
    zones:
      - zone-a
      - zone-b
      - zone-c
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
---
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: worker
spec:
  virtualMachine:
    cpu:
      cores: 4
      coreFraction: 100%
    memory:
      size: 8Gi
    virtualMachineClassName: generic
  rootDisk:
    size: 10Gi
    image:
      kind: ClusterVirtualImage
      name: ubuntu-2204
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: worker
    maxPerZone: 1
    minPerZone: 1
    zones:
      - zone-a
      - zone-b
      - zone-c
```

Replacing the OS image

The `rootDisk.image` parameter of DVPInstanceClass defines the image for `CloudEphemeral` nodes. With a ModuleConfig-based provider configuration, this parameter also defines the image for `CloudPermanent` nodes and master nodes.

With a DVPClusterConfiguration-based configuration, the image for `CloudPermanent` nodes and master nodes is set by `rootDisk.image` in `nodeGroups[].instanceClass` and `masterNodeGroup.instanceClass`, respectively.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

Additional resources

- Global custom resources
- Module `cloud-provider-dvp` custom resources
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in VMware Cloud Director

Resource preparation

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To manage resources in VCD using the “Deckhouse Platform”, the following resources must be configured in the system:

- Organization
- VirtualDataCenter
- vApp (for the “Standard” placement scheme)
- StoragePolicy
- SizingPolicy
- Network (for the “Standard” placement scheme)
- EdgeRouter
- Catalog

The Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter, and Catalog resources must be provided by your VMware Cloud Director service provider.

The Network (internal network) can be configured by your VMware Cloud Director service provider or by yourself. When using the “StandardWithNetwork” placement scheme, the network is created automatically. Below is a method for manually setting up an internal network.

User permissions

For access to the VMware Cloud Director API, one of the following options is sufficient:

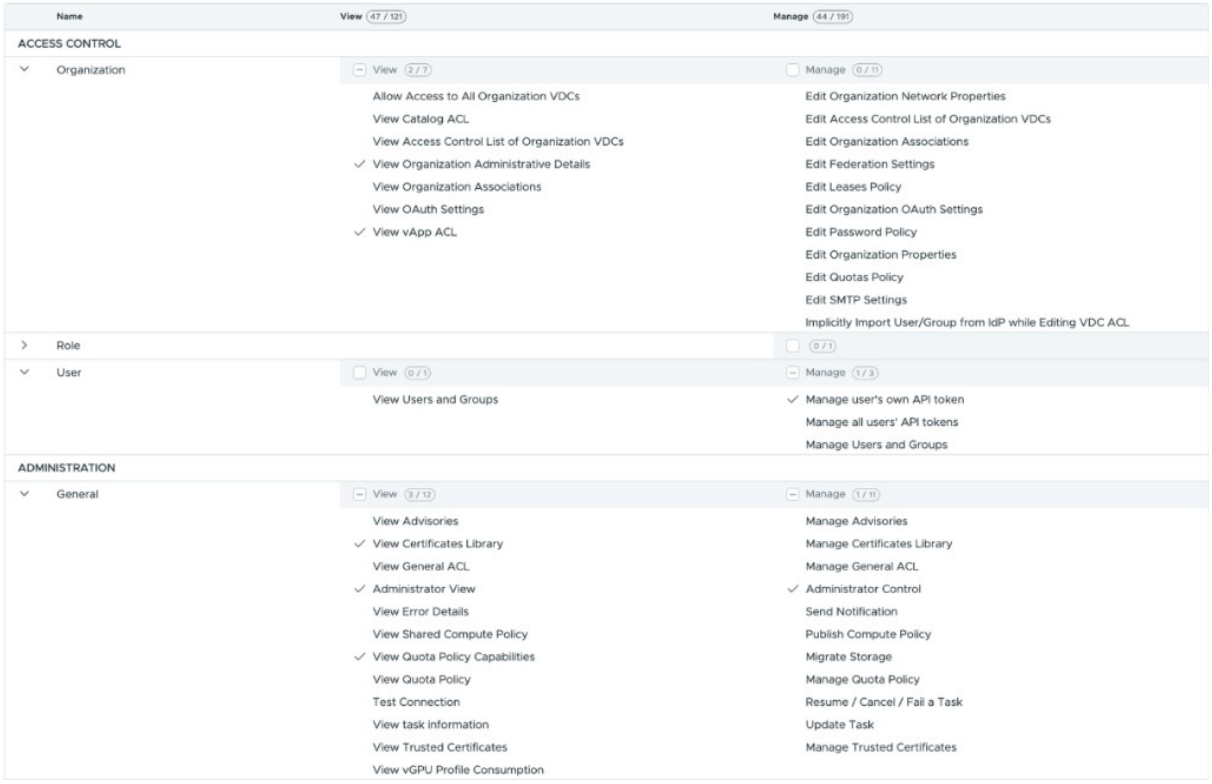
- The `Organization Administrator` role with the `Preserve All ExtraConfig Elements During OVF Import and Export` right. This option does not require configuring a separate role, but grants the user full administrative access to the organization.
- A custom role with the set of rights listed below. This set is sufficient to create a cluster, provision nodes (including CloudEphemeral with static addressing and DHCP), and work with CSI disks and StorageClass without granting additional administrative rights.

In VMware Cloud Director, user permissions come from the role and from a **Rights Bundle** — a set of rights published to the organization. Add the `Preserve All ExtraConfig Elements During OVF Import and Export` right both to the role and to the organization’s Rights Bundle.

> ⚠ Without the `Preserve All ExtraConfig Elements During OVF Import and Export` right in both the role and the organization’s Rights Bundle, `userdata` is not passed to the virtual machines of ephemeral nodes.

To create a custom role, enable the rights as shown below. Right names match the VCD UI.

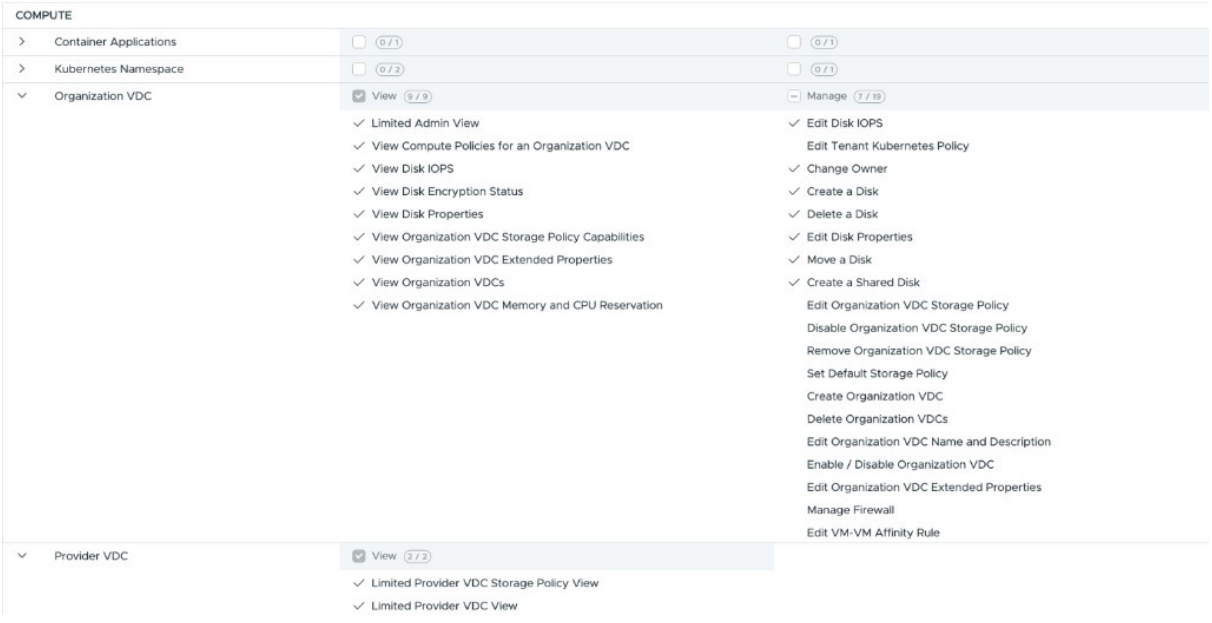
1. In the “ACCESS CONTROL” and “ADMINISTRATION” sections, enable the following rights:
   - Organization → View: `View Organization Administrative Details`, `View vApp ACL`
   - User → Manage: `Manage user's own API token`
   - General → View: `View Certificates Library`, `Administrator View`, `View Quota Policy Capabilities`
   - General → Manage: `Administrator Control`



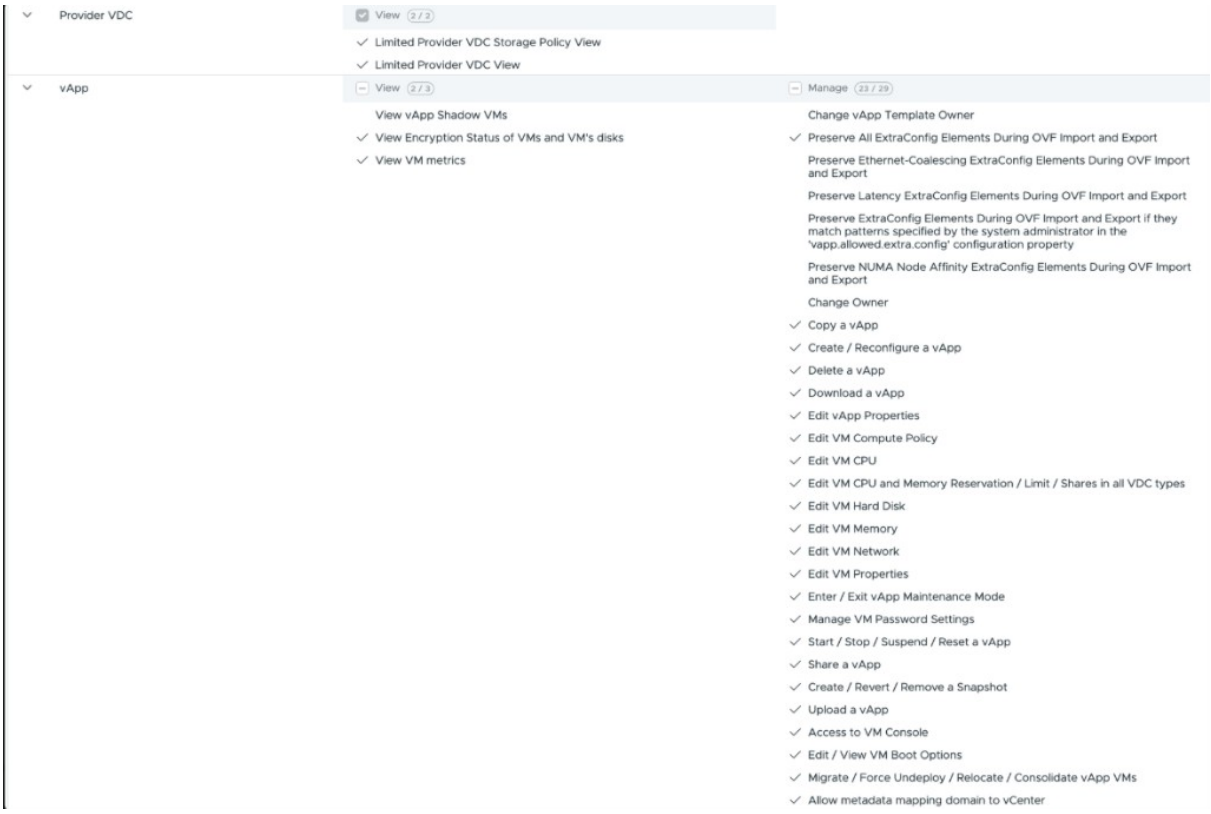
2. In the “COMPUTE” → “Organization VDC” section, enable all View rights and the following Manage rights:

   - `Edit Disk IOPS`
   - `Edit Tenant Kubernetes Policy`
   - `Change Owner`
   - `Create a Disk`
   - `Delete a Disk`
   - `Edit Disk Properties`
   - `Move a Disk`
   - `Create a Shared Disk`

   In the “Provider VDC” → View section, enable `Limited Provider VDC Storage Policy View` and `Limited Provider VDC View`:



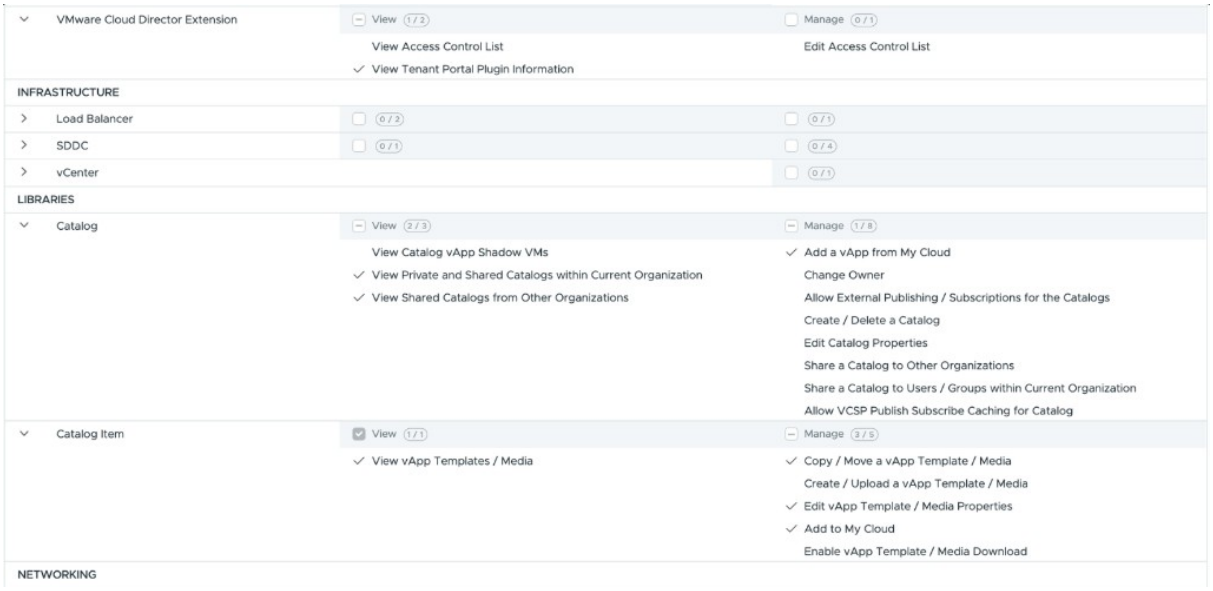
3. In the “COMPUTE” → “vApp” section, enable the rights as shown in the screenshot. Make sure that `Preserve All ExtraConfig Elements During OVF Import and Export` is enabled. In the verified role, View includes `View Encryption Status of VMs and VM's disks` and `View VM metrics`; `View vApp Shadow VMs` and the other Preserve ExtraConfig options are not enabled:



4. In the “LIBRARIES” section, enable the following rights:

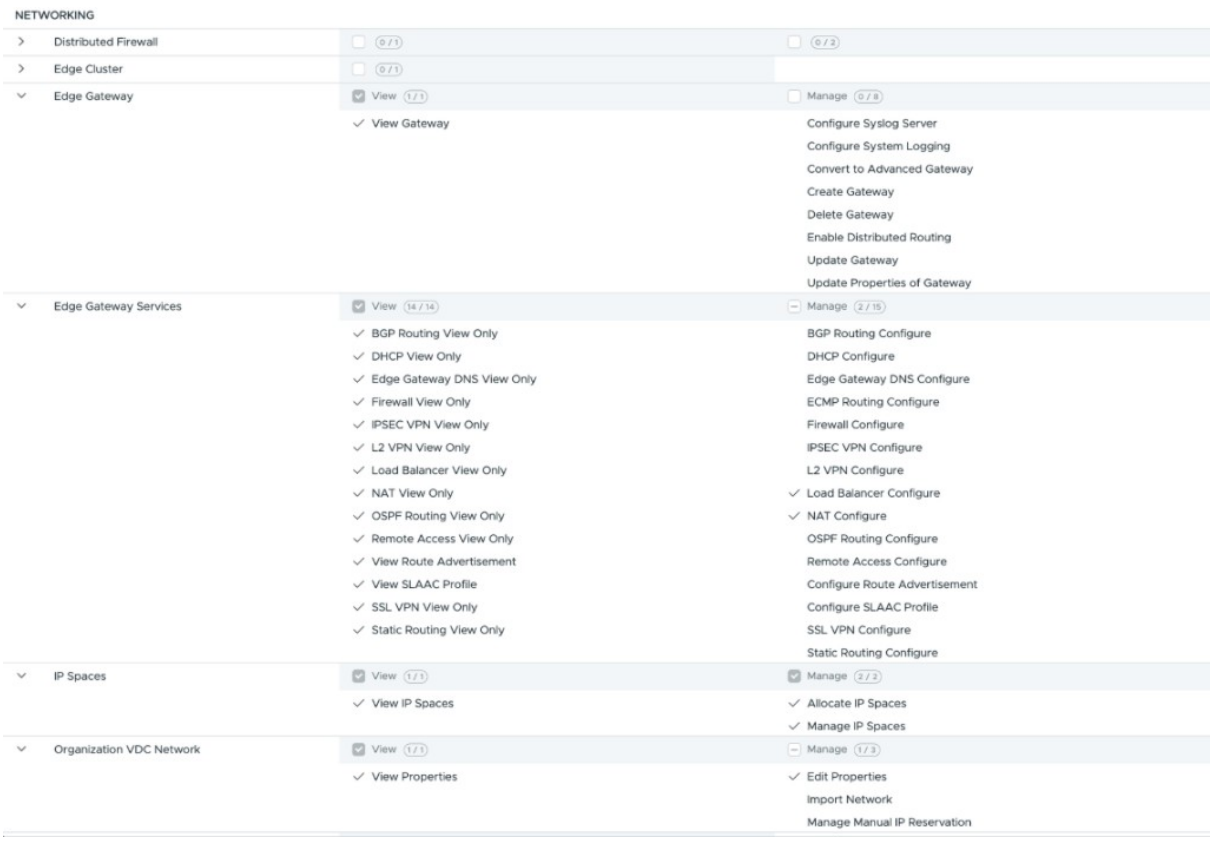
   - Catalog → View: `View Private and Shared Catalogs within Current Organization`, `View Shared Catalogs from Other Organizations` (to use images from shared catalogs)
   - Catalog → Manage: `Add a vApp from My Cloud`
   - Catalog Item → View: `View vApp Templates / Media`
   - Catalog Item → Manage: `Copy / Move a vApp Template / Media`, `Edit vApp Template / Media Properties`, `Add to My Cloud`

   In the “VMware Cloud Director Extension” → View section, enable `View Tenant Portal Plugin Information`:

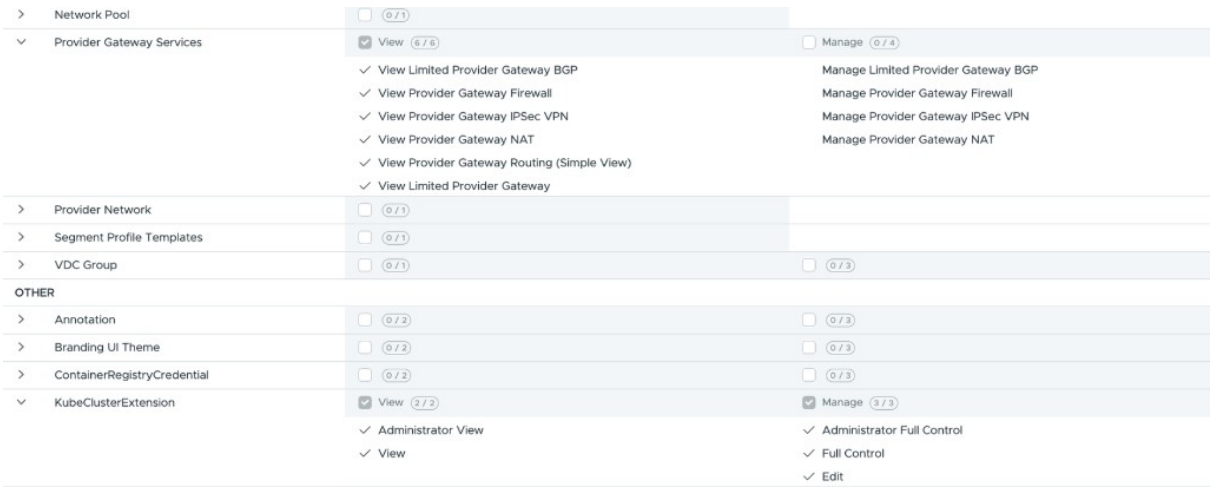


5. In the “NETWORKING” section, enable the following rights:

   - Edge Gateway → View: `View Gateway`
   - Edge Gateway Services → View: all items
   - Edge Gateway Services → Manage: `Load Balancer Configure`, `NAT Configure`
   - IP Spaces → View: `View IP Spaces`
   - IP Spaces → Manage: `Allocate IP Spaces`, `Manage IP Spaces`
   - Organization VDC Network → View: `View Properties`
   - Organization VDC Network → Manage: `Edit Properties` (required to create networks in the organization)



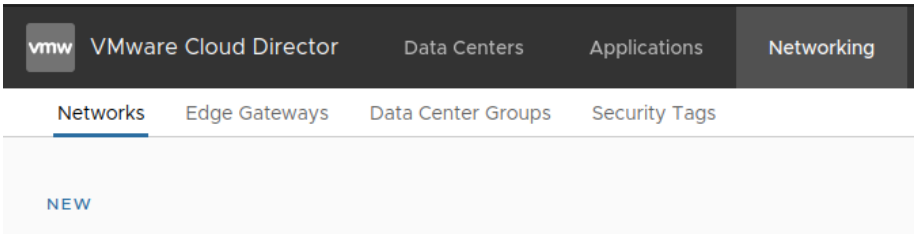
6. In the “Provider Gateway Services” section, enable all View rights. In the “KubeClusterExtension” section, enable all View and Manage rights:



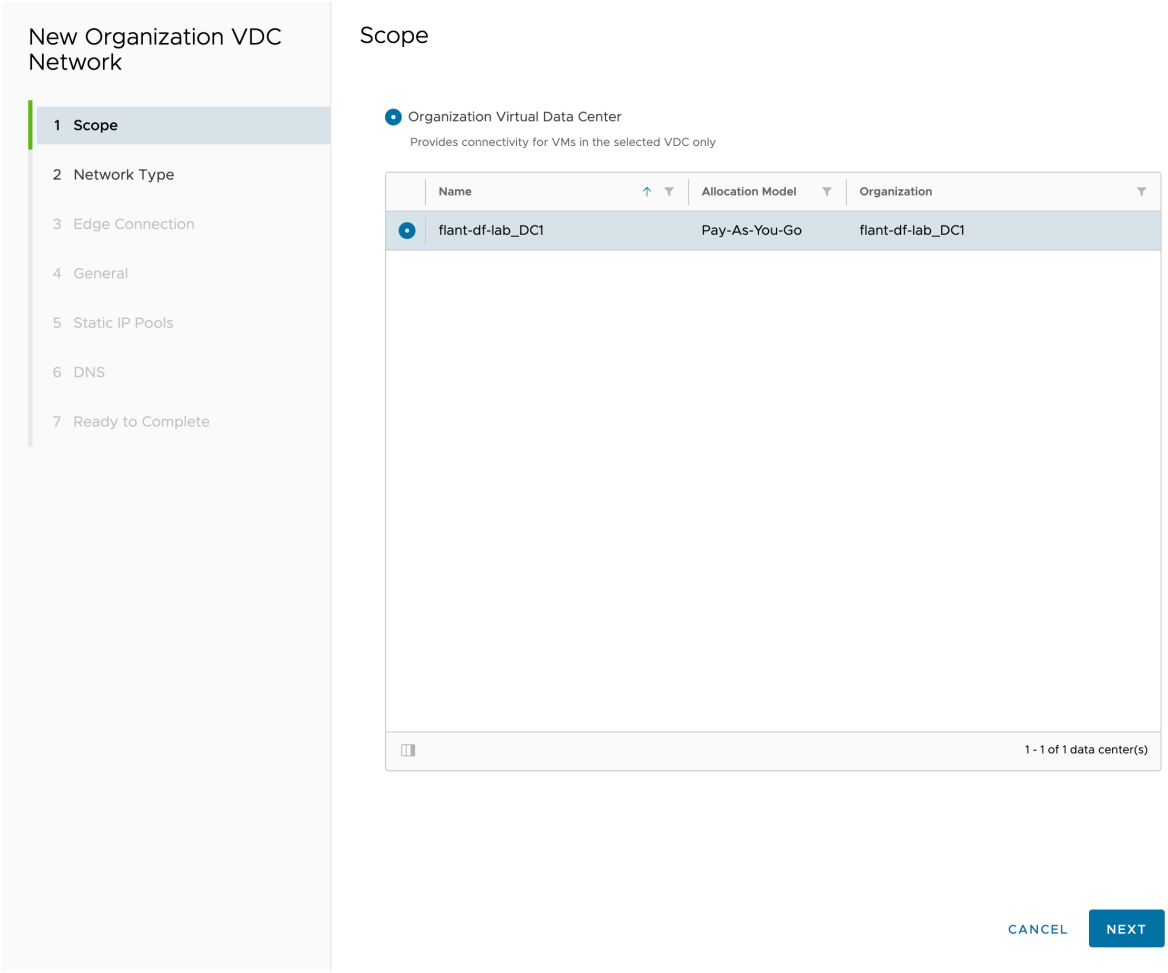
Adding a network

> **i** This instruction applies only to the “Standard” placement scheme.

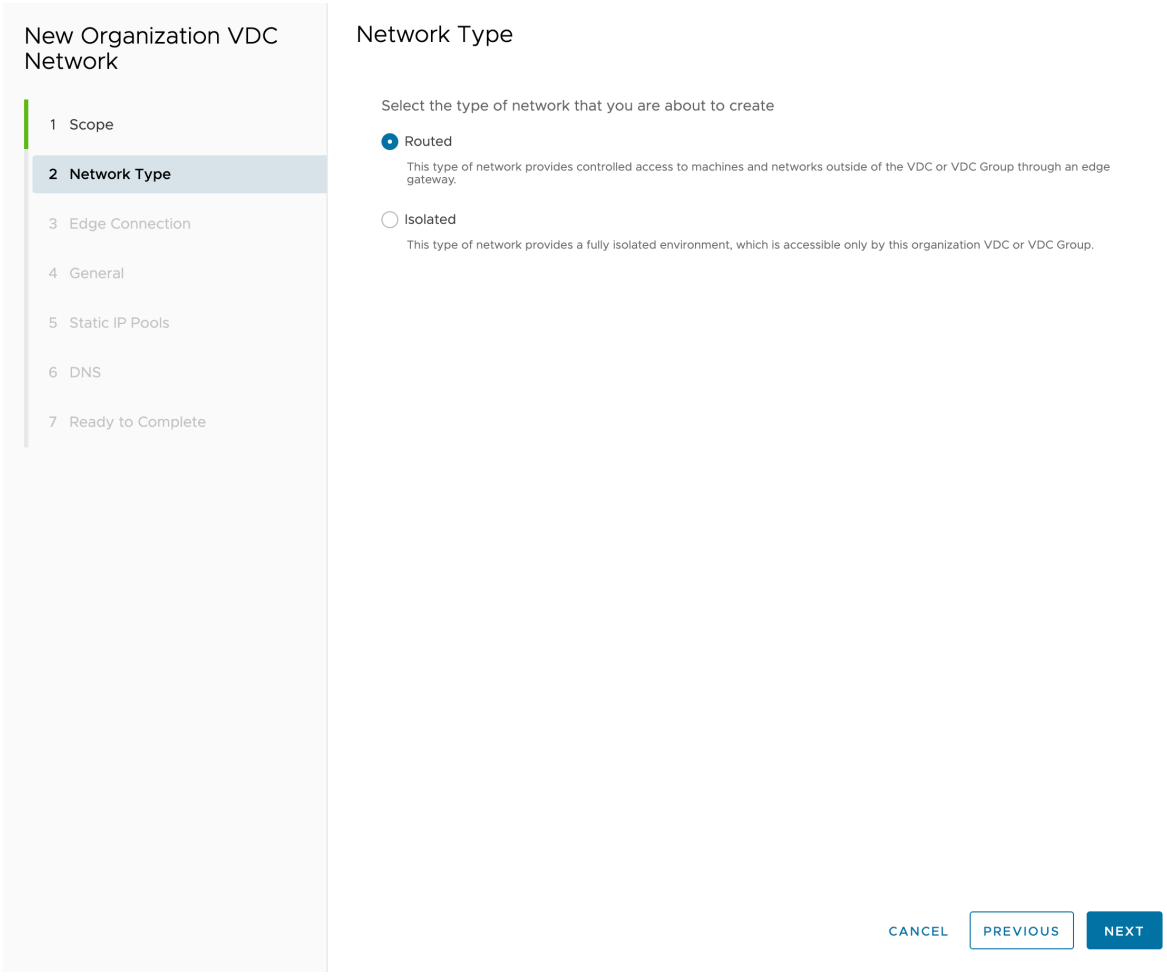
1. Go to the “Networking” tab and click “NEW”:



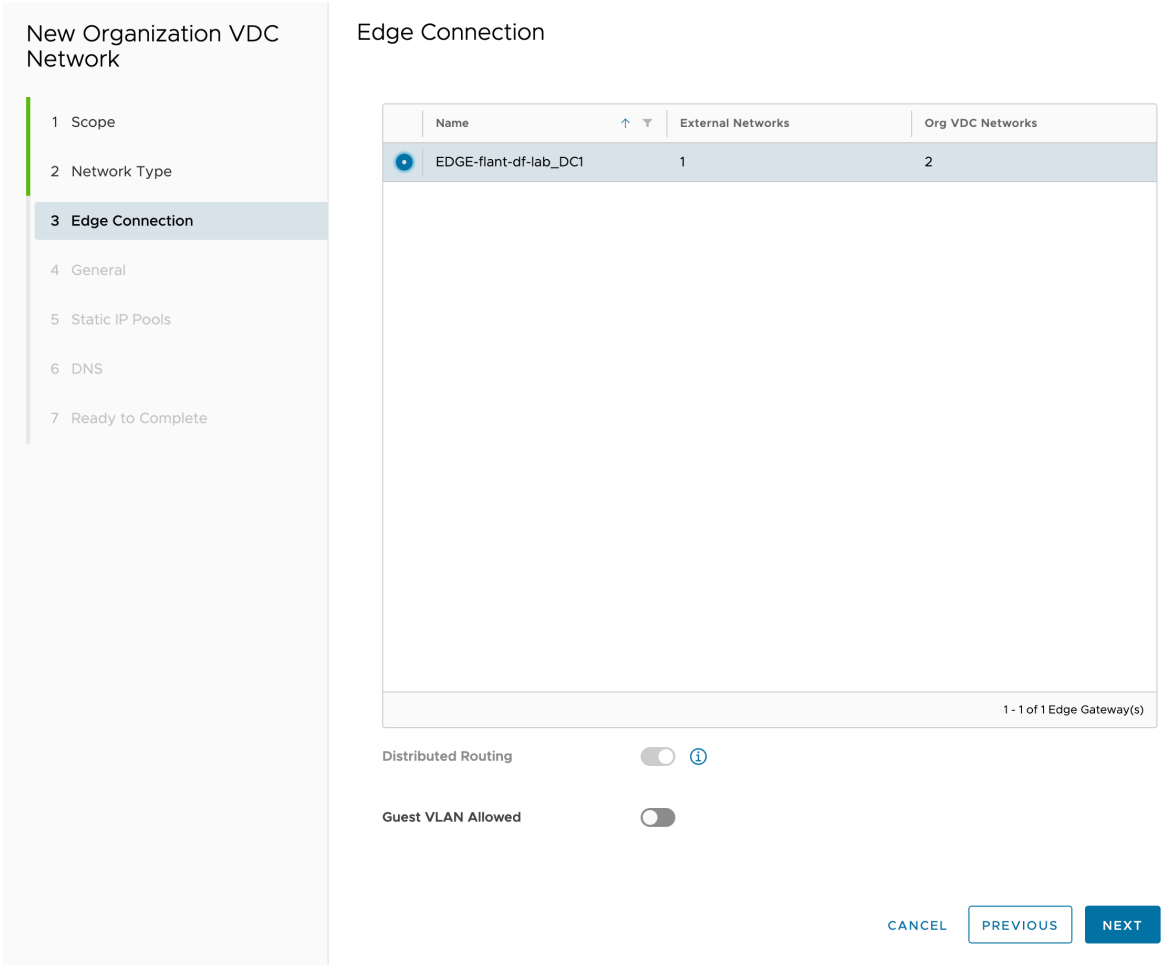
2. Select the desired “Data Center”:



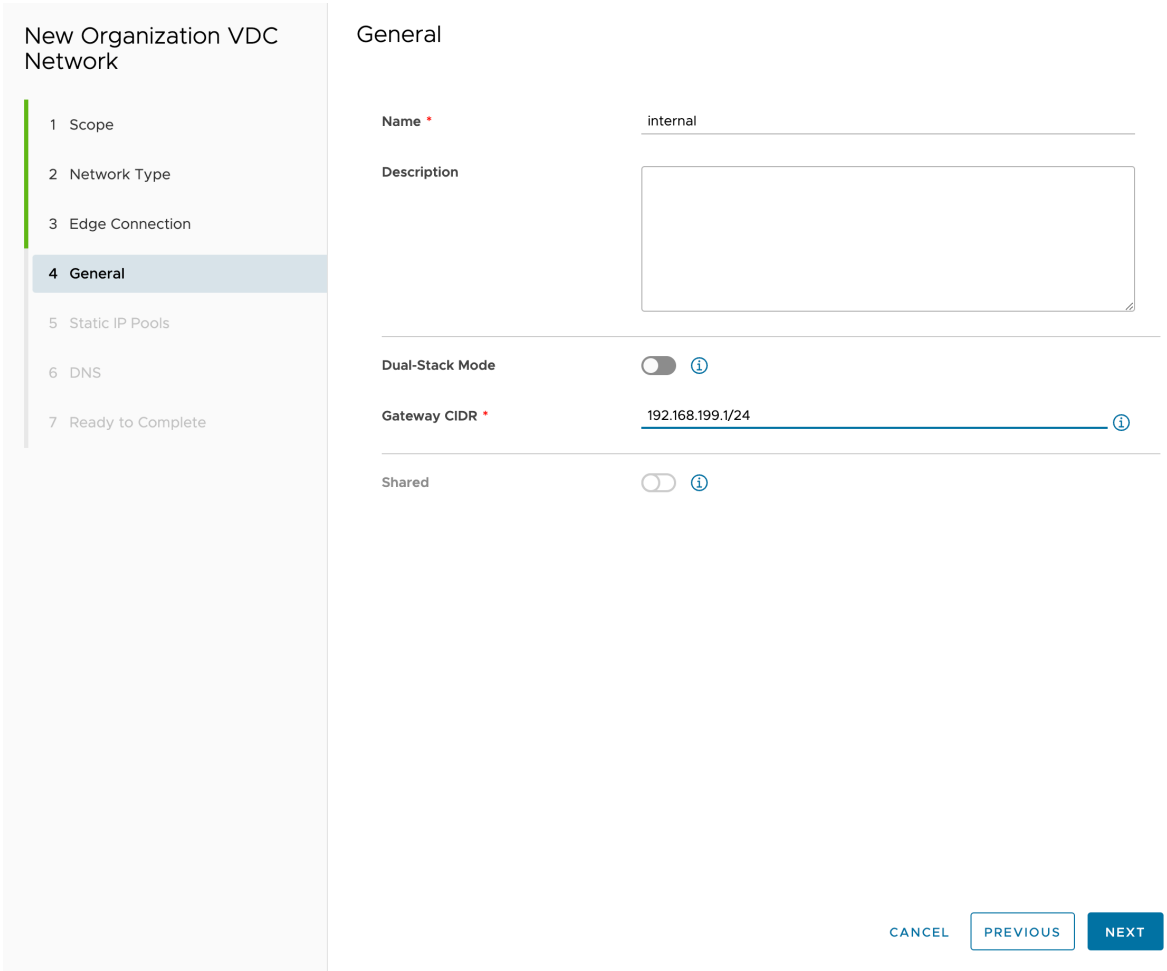
3. In the “Network type” step, select “Routed”:



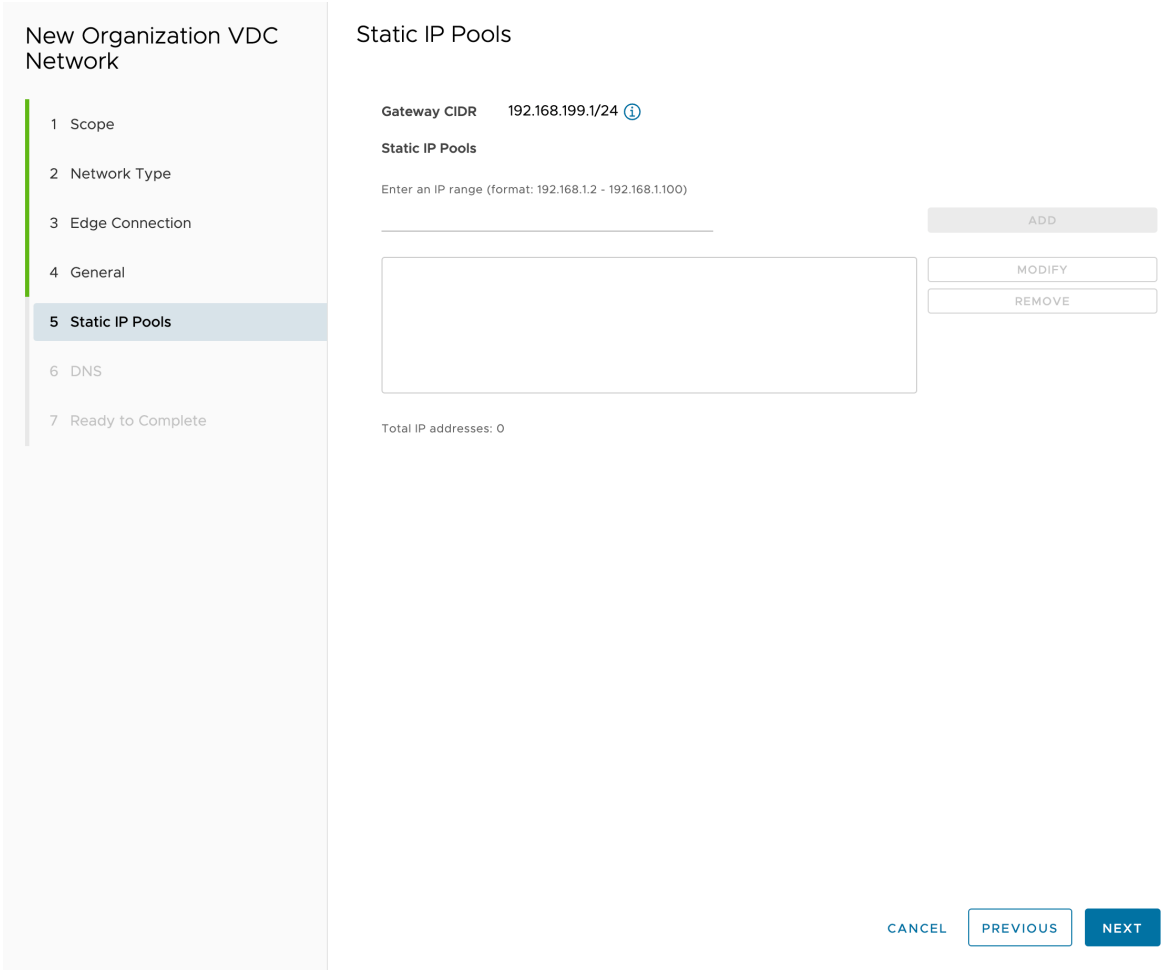
4. Connect the “EdgeRouter” to the network:



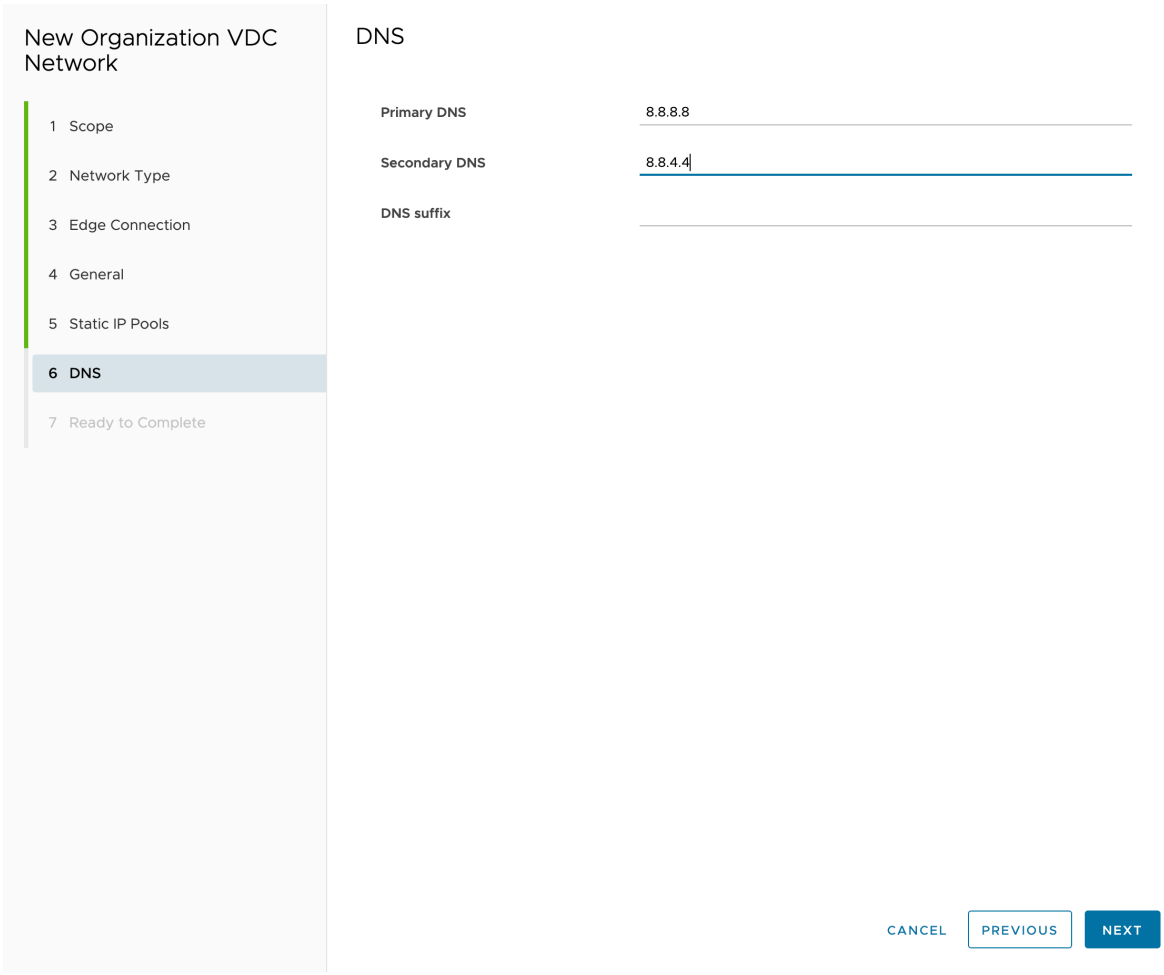
5. Specify the network name and CIDR:



6. Do not add “Static IP Pools” since DHCP will be used:



7. Specify DNS server addresses:



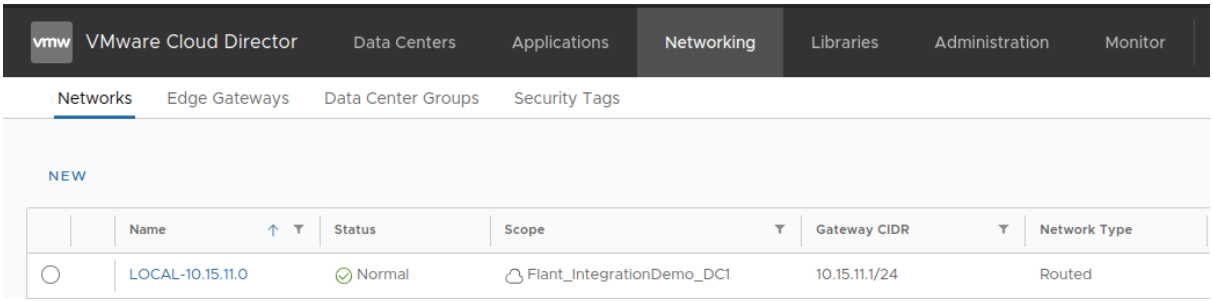
DHCP setup

> **i** This instruction applies only to the “Standard” placement scheme.

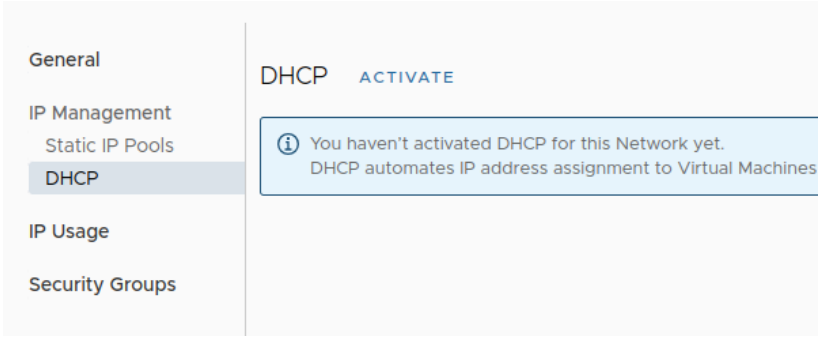
To dynamically provision nodes, enable the DHCP server for the internal network.

> **i** We recommend reserving the beginning of the address range for system workloads (control plane, frontend nodes, system nodes), and using the rest for the DHCP pool.
> For example, for a “/24” network, reserving 20 addresses for system workloads is sufficient.

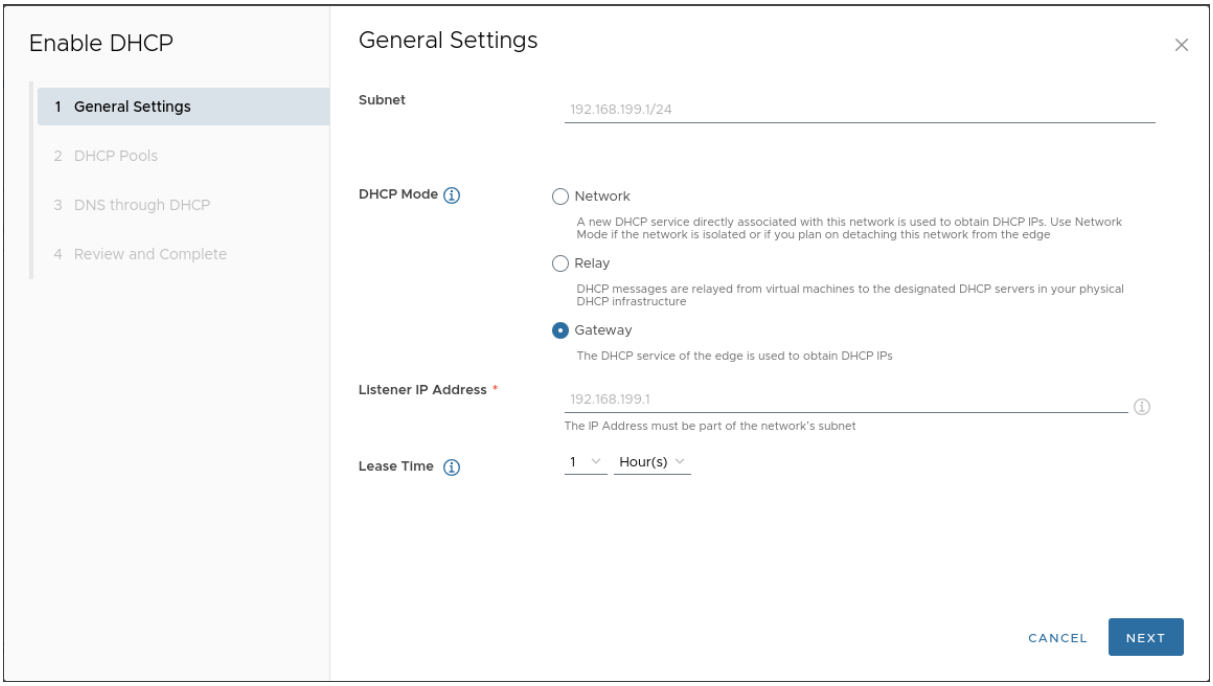
1. Go to the “Networking” tab and open the created network:



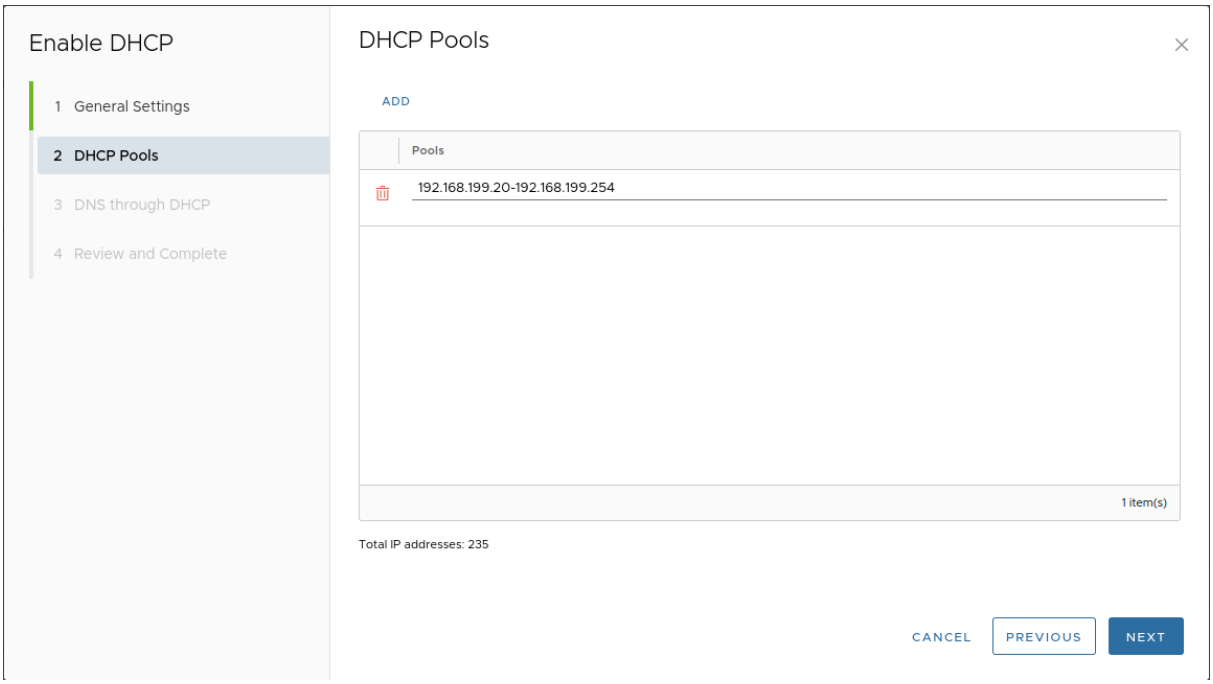
2. In the opened window, select “IP Management” → “DHCP” → “Activate”:



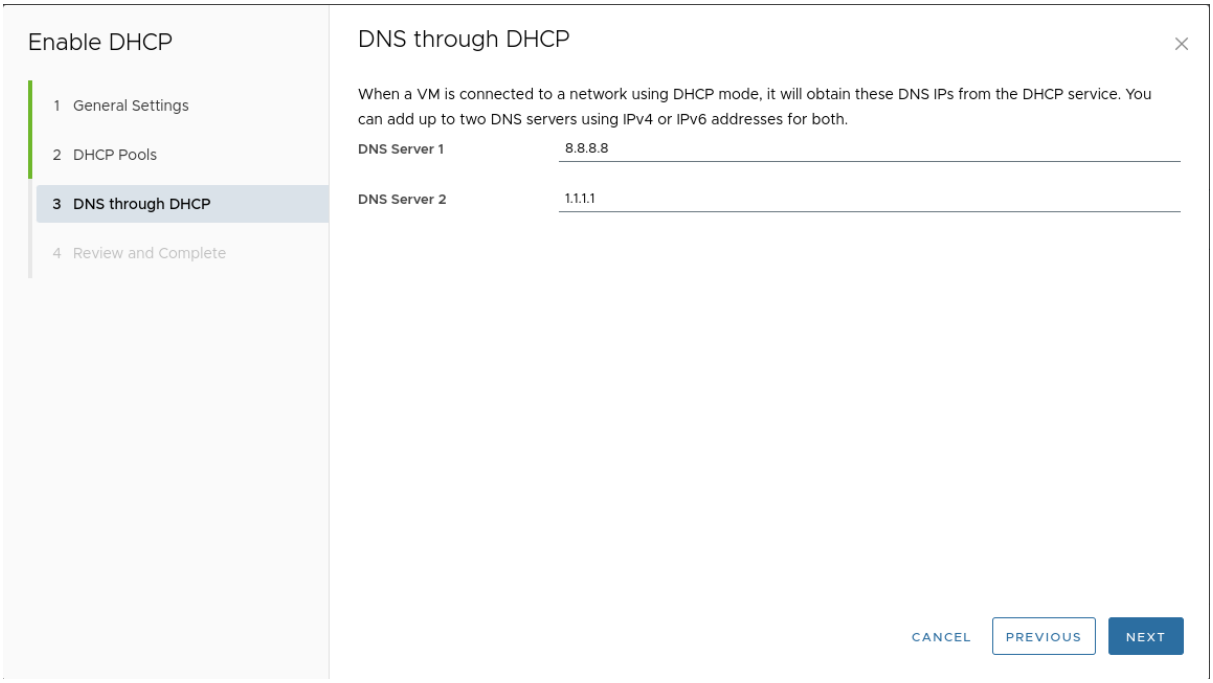
3. In the “General settings” tab, configure parameters as shown in the example:



4. Add a pool:



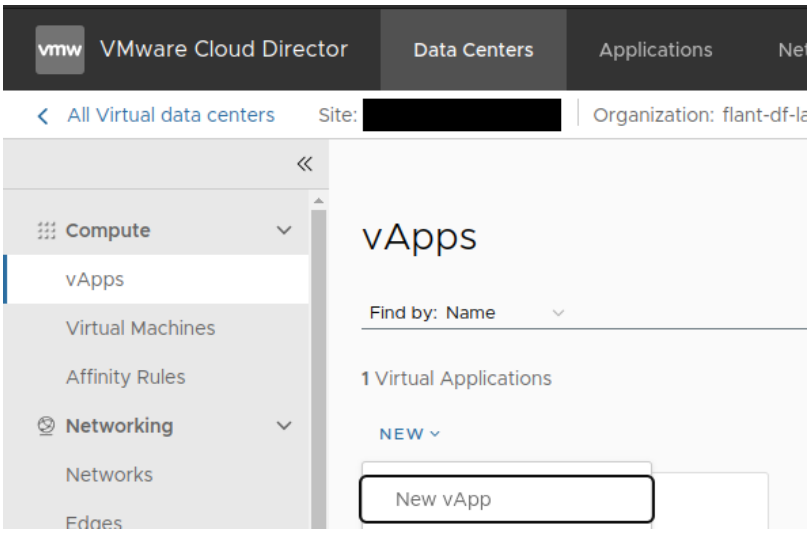
5. Specify DNS server addresses:



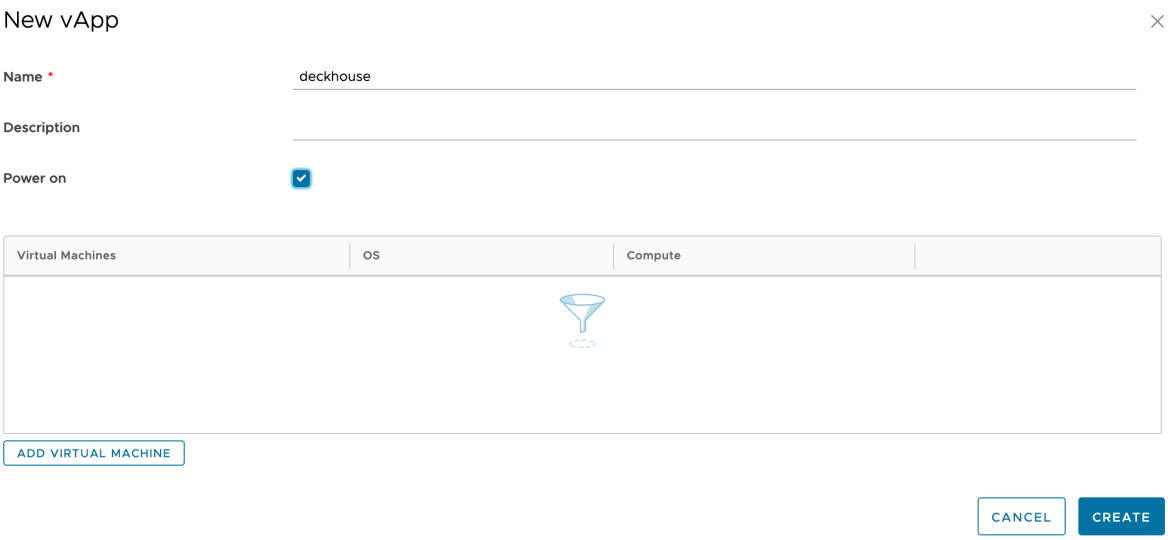
Adding a vApp

> **i** This instruction applies only to the “Standard” placement scheme.

1. Go to “Data Centers” → “vApps” → “NEW” → “New vApp”:



2. Specify a name and enable the vApp:

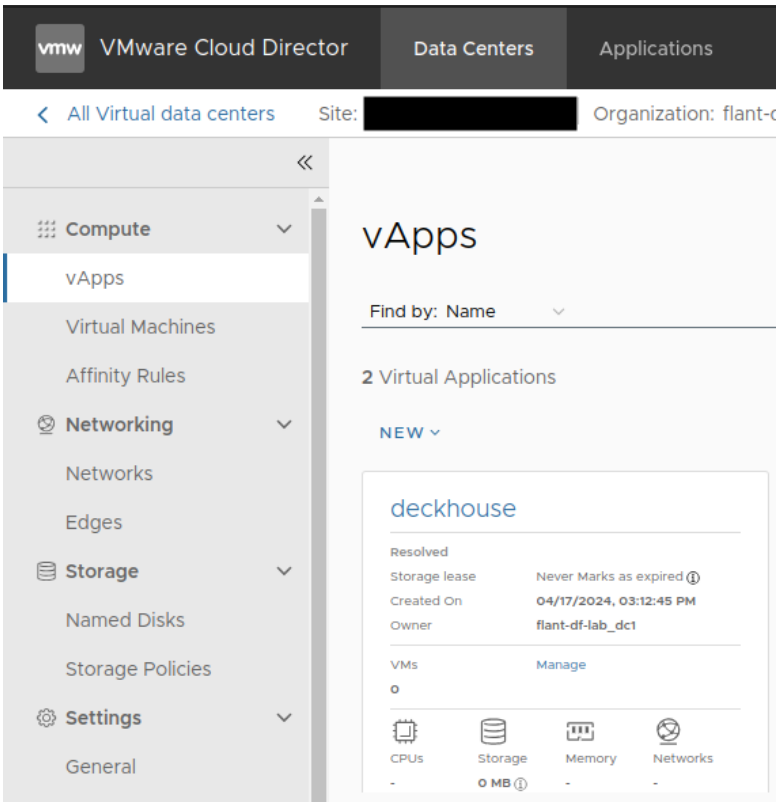


Adding a network to a vApp

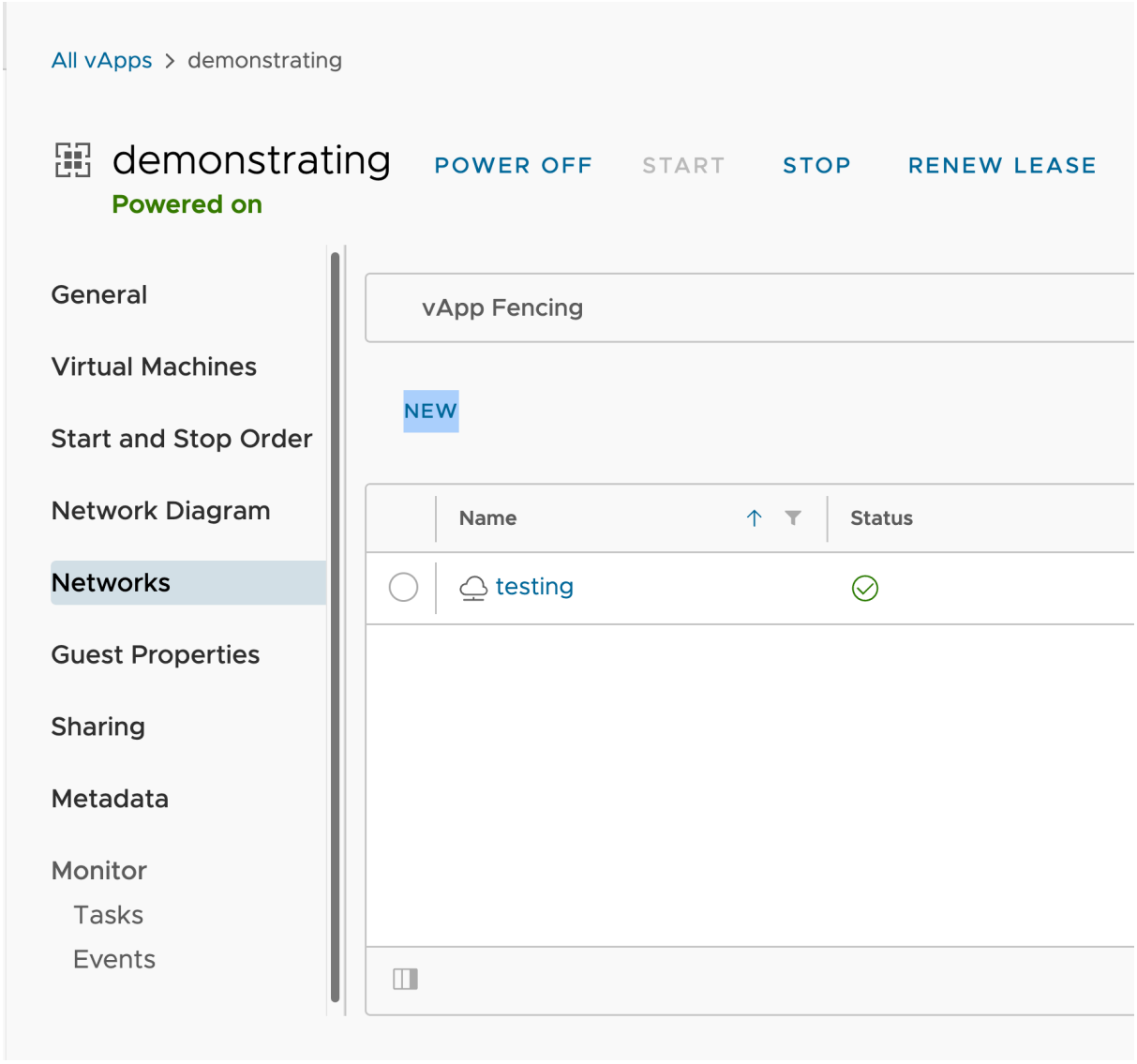
> **i** This instruction applies only to the “Standard” placement scheme.

After creating the vApp, attach the created internal network to it.

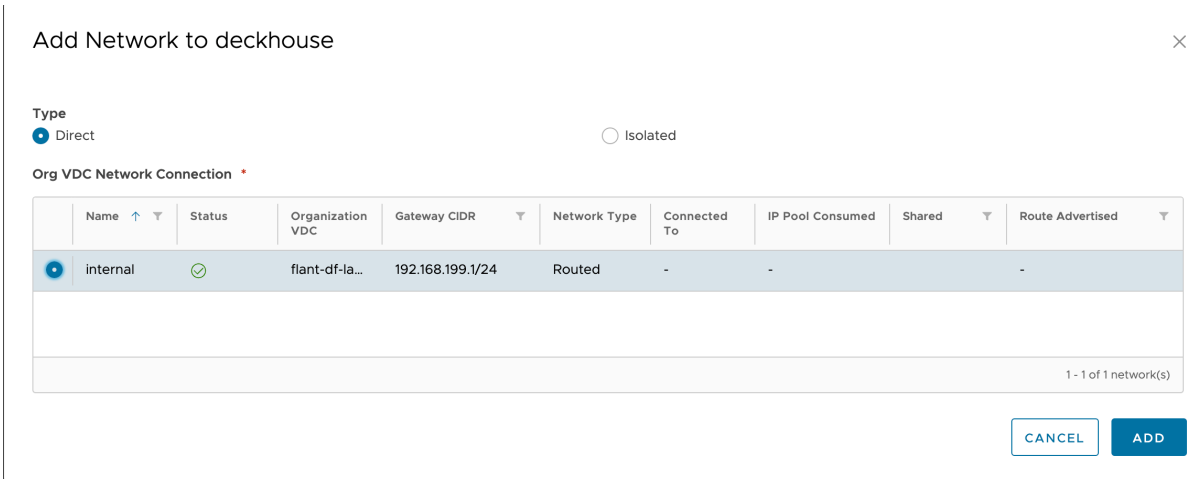
1. Go to “Data Centers” → “vApps” and open the desired vApp:



2. Go to the “Networks” tab and click “NEW”:



3. In the pop-up window, select “Direct” type and choose the network:

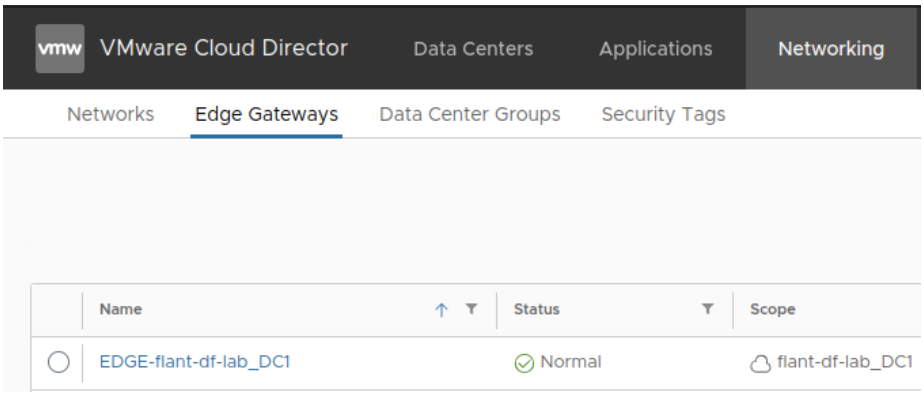


Incoming traffic

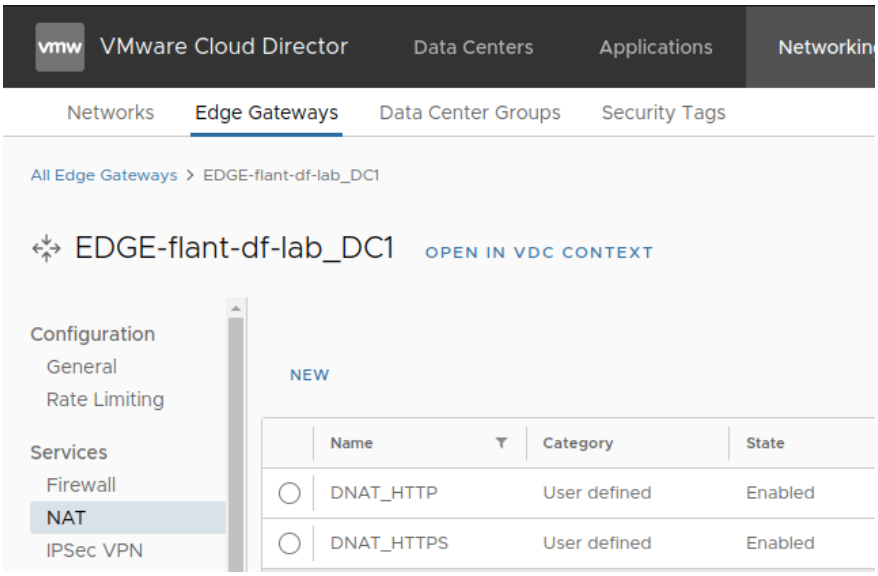
Incoming traffic must be directed to the edge router (ports “80”, “443”) using DNAT rules to the allocated address in the internal network.
This address is managed by MetalLB in L2 mode on dedicated frontend nodes.

Configuring DNAT/SNAT rules on the Edge Gateway

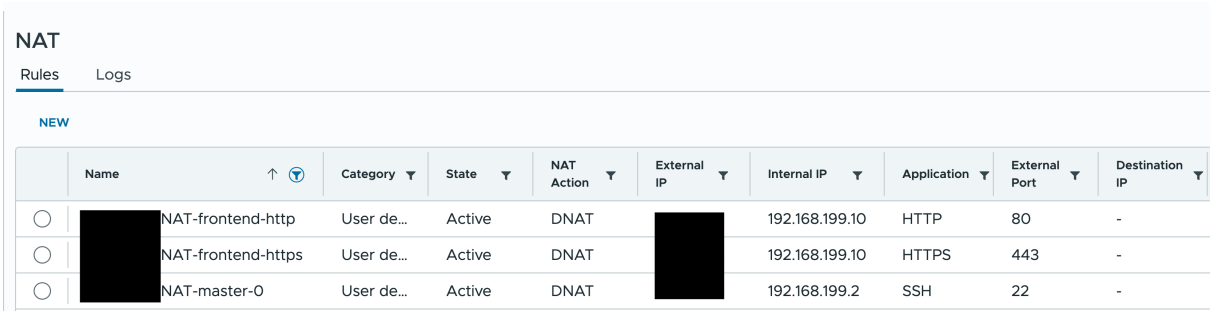
1. Go to “Networking” → “Edge Gateways”, open the edge gateway:



2. Go to “Services” → “NAT”:



3. Add the following rules:



The first two rules are for incoming traffic, and the third is for SSH access to the control plane node (required for installation).

4. To allow virtual machines to access the internet, configure SNAT rules as in the example:



This rule allows VMs from the “192.168.199.0/24” subnet to access the internet.

Let’s Encrypt certificate issuance

> **i** This instruction is applicable only for the `Standard` deployment layout.

When using `cert-manager` with the HTTP-01 ACME challenge type, you may need to configure Hairpin NAT (NAT loopback) on the Edge Gateway.

This is required when the Ingress domain name resolves to the external IP address of the Edge Gateway, while requests to that address are made from **inside** the cluster (from the node network). In this case, the traffic must be correctly routed back to the internal network to the Ingress address (for example, the MetalLB address).

This is important for issuing Let’s Encrypt certificates because cert-manager performs a preliminary self-check of the challenge URL before sending a request to the ACME provider. If the URL is not reachable from inside the cluster, certificate issuance will not start, even if the address is accessible from outside.

Example (cluster network `192.168.199.0/24`):

- internal node network: `192.168.199.0/24`
- external Edge Gateway IP address: `194.117.83.19`
- internal Ingress address (for example, MetalLB): `192.168.199.251`.

In this case, you must configure Hairpin NAT for traffic from the `192.168.199.0/24` network that is sent to the external IP `194.117.83.19`, with translation to the internal Ingress address `192.168.199.251`. Apply Hairpin NAT rules (DNAT and SNAT if needed) to the **local interface** of the Edge Gateway, not to the Uplink.

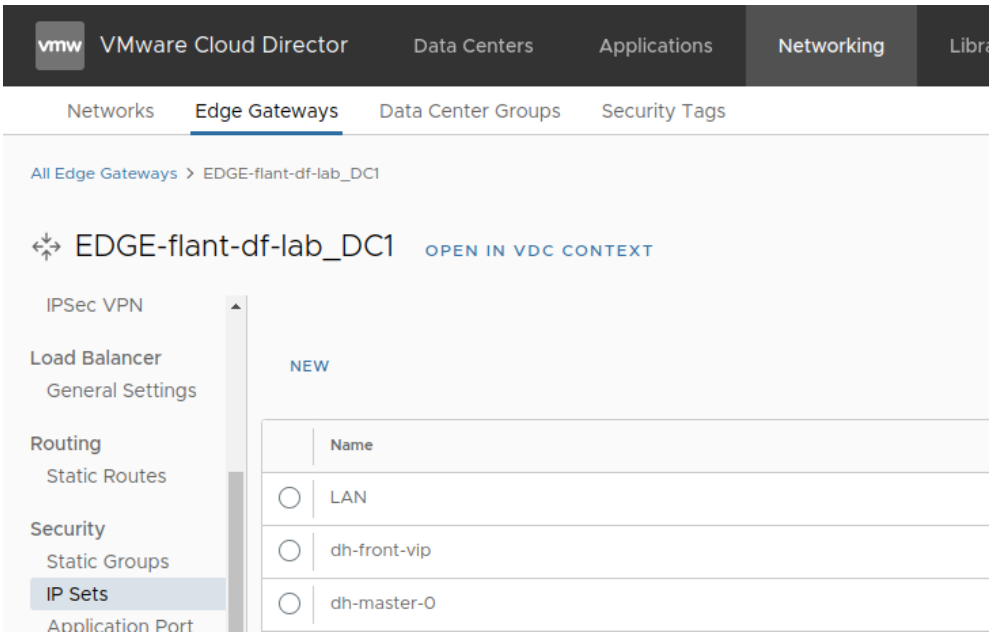
> **i** Hairpin NAT configuration may be required for Edge Gateway based on NSX-V.
>
> For NSX-T, the loopback scenario is often supported by default, but the actual behavior depends on the VMware Cloud Director configuration.

Firewall setup

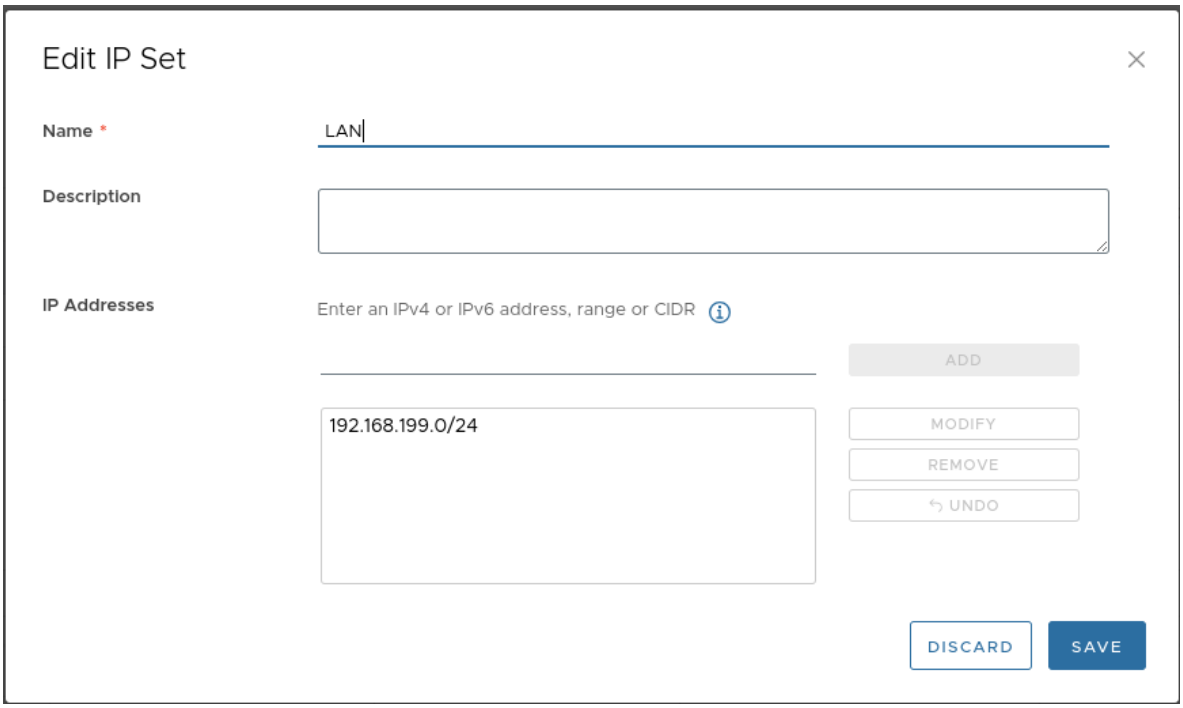
> **i** This instruction applies only to the “Standard” placement scheme.

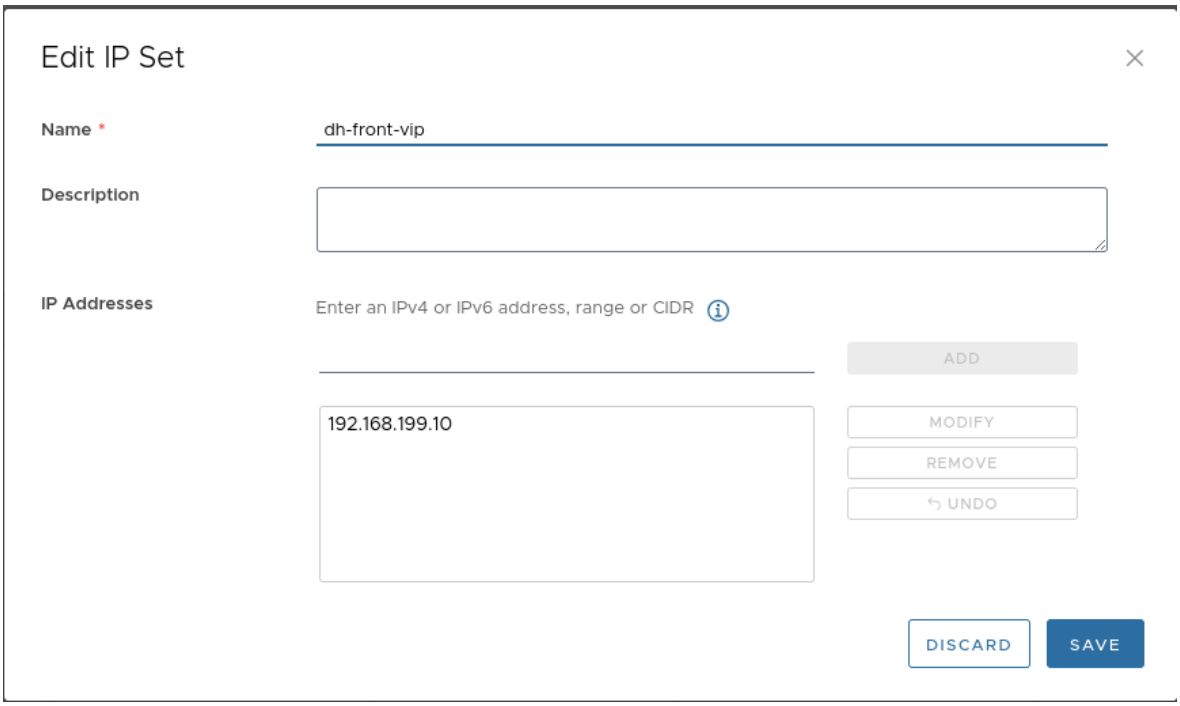
After configuring DNAT, configure the firewall. Start by setting up IP sets.

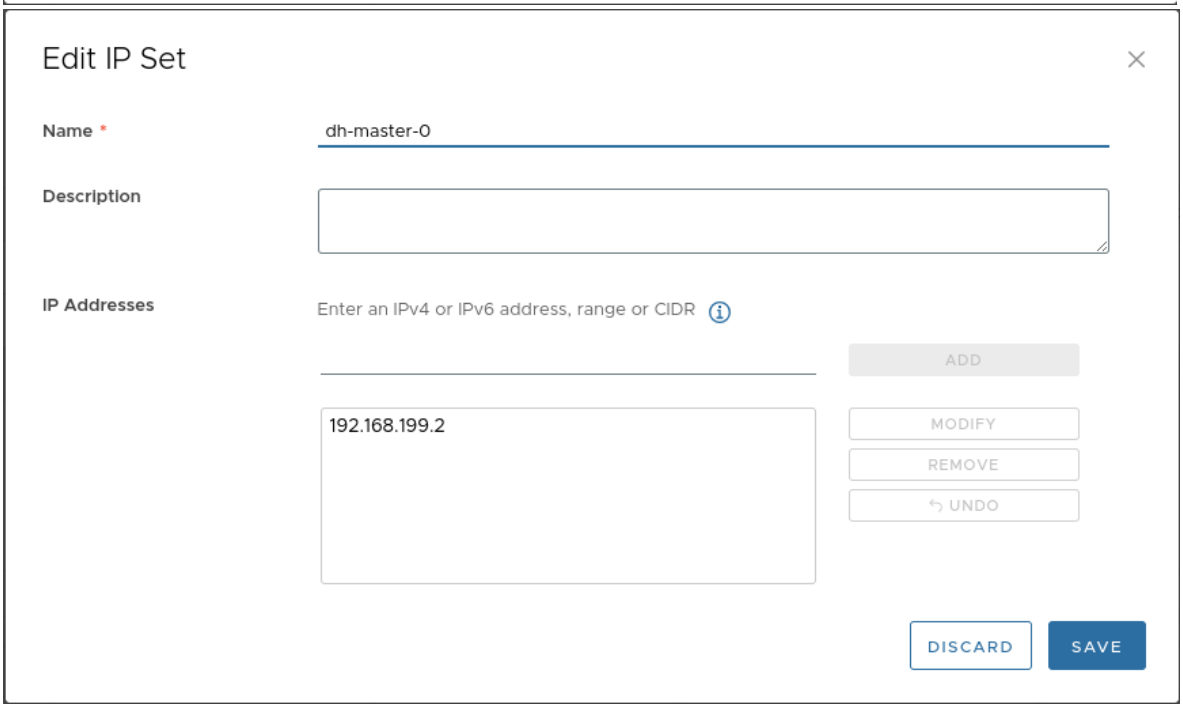
1. Go to “Security” → “IP Sets”:



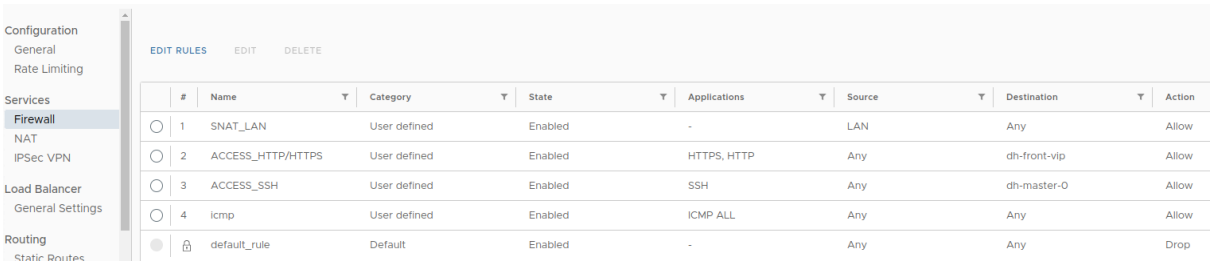
2. Create the following IP set (assuming the MetalLB address will be “.10” and the control plane node “.2”):







3. Add the following firewall rules:



Virtual machine template

⚠ The provider has been tested only with virtual machine templates based on “Ubuntu 22.04”.

⚠ Disable vApp/Guest OS Customization (Guest Customization, vApp Customization, and similar mechanisms) for the cluster template and virtual machines in VMware Cloud Director. DP performs the initial node provisioning via `cloud-init` (OVF/VMware GuestInfo datasource). If customization is enabled, it may conflict with `cloud-init` and result in incorrect node initialization.

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

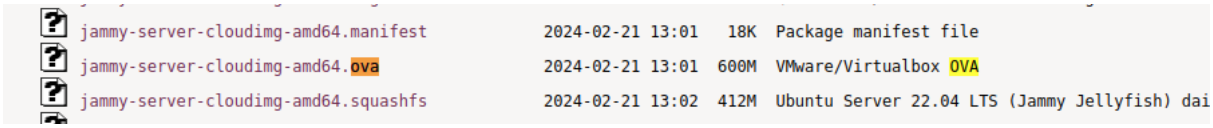
⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

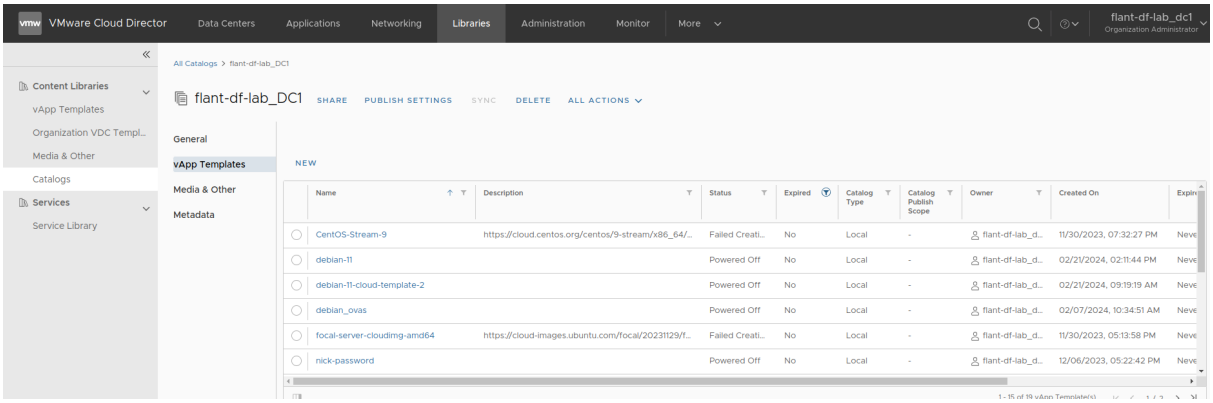
In the example, an “OVA” file provided by Ubuntu is used, with two modifications.
These modifications are required for proper provisioning of CloudPermanent nodes and to allow attaching disks created by CSI.

Preparing the template from an OVA file

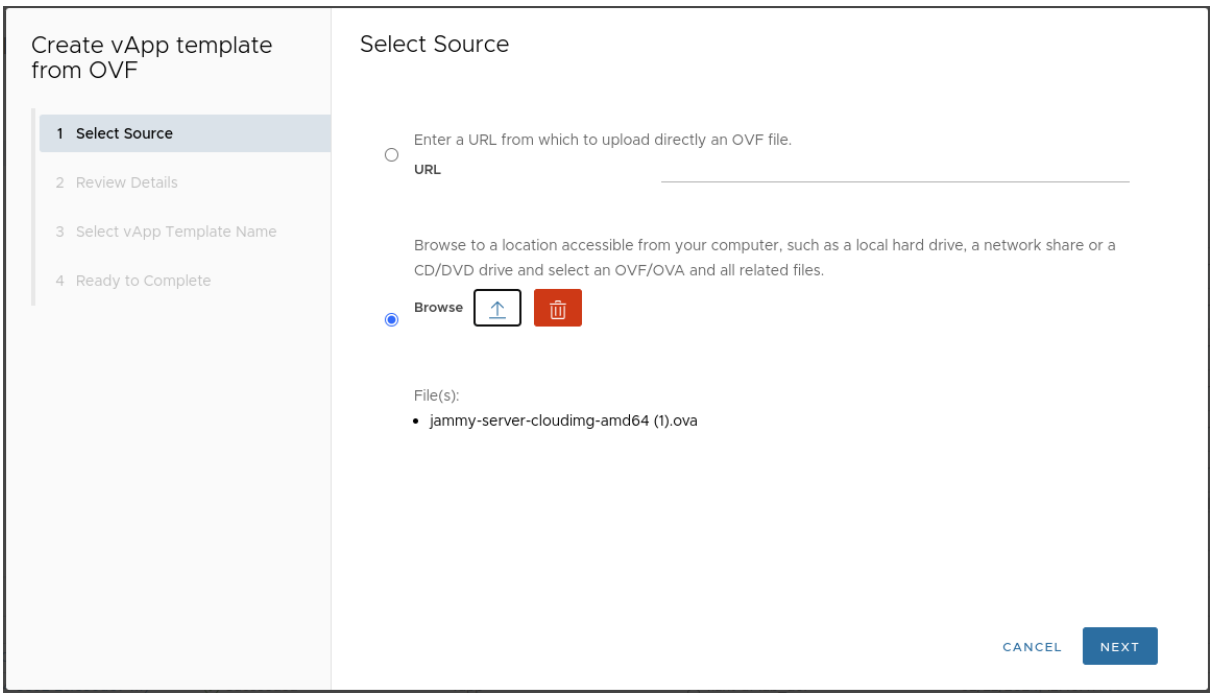
1. Download the OVA file:

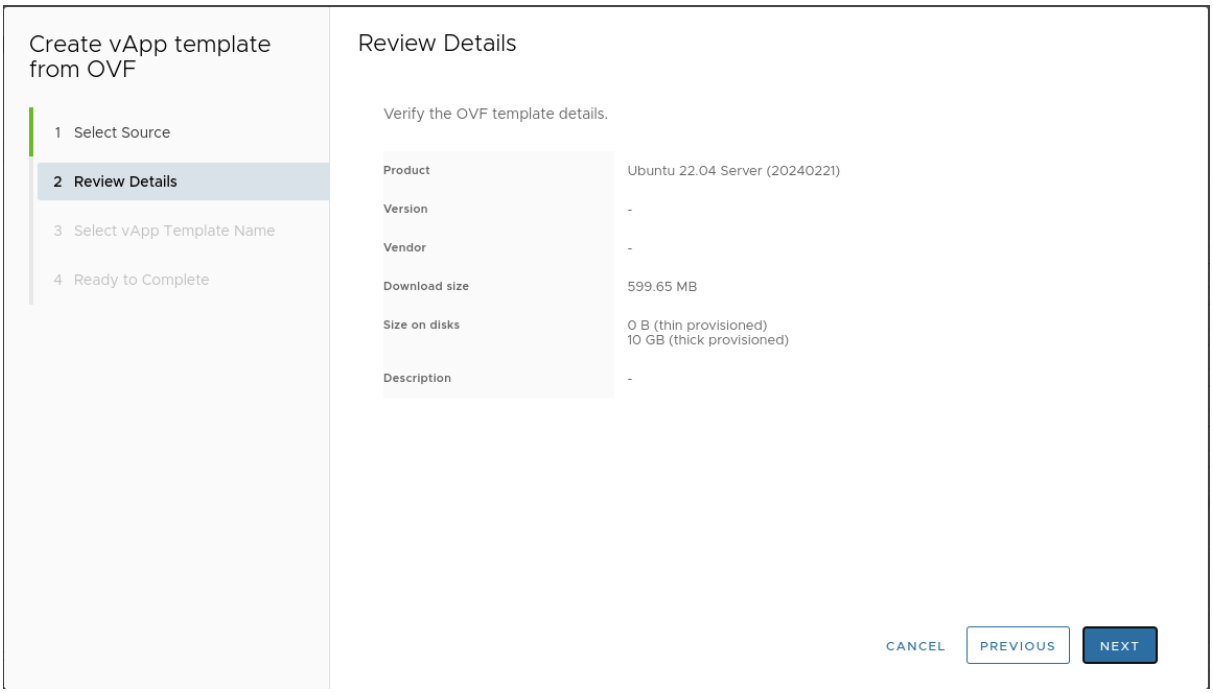


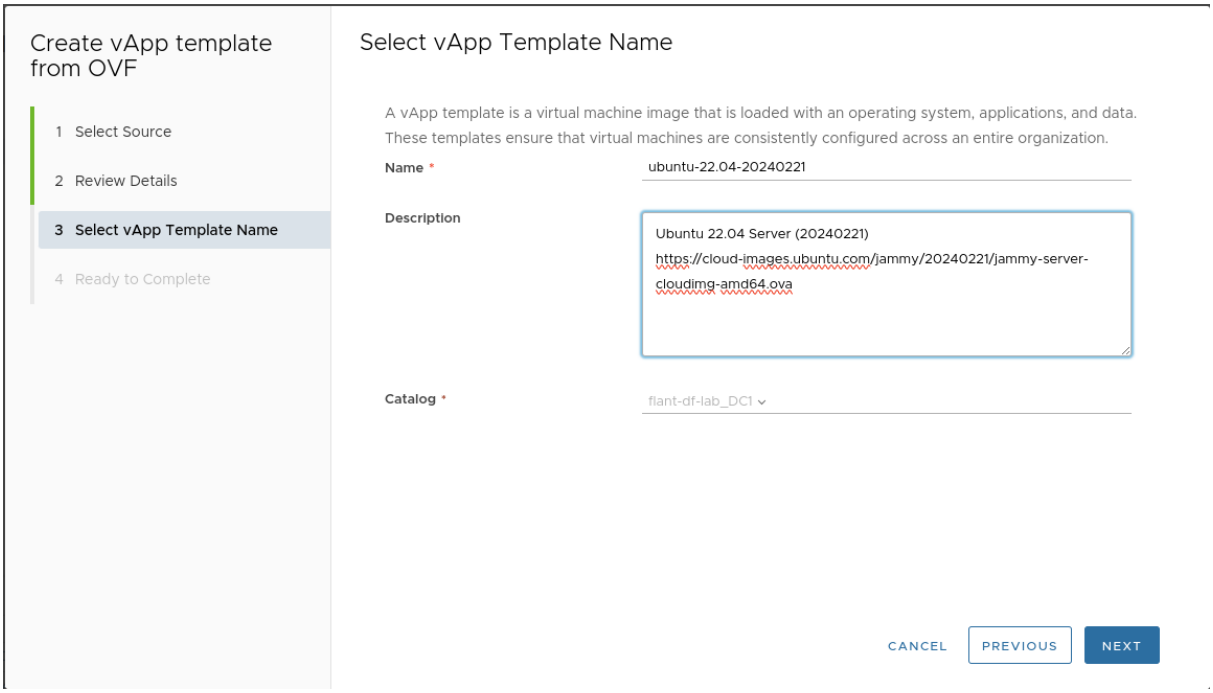
2. Go to “Libraries” → “Catalogs” → “Organization Catalog”:



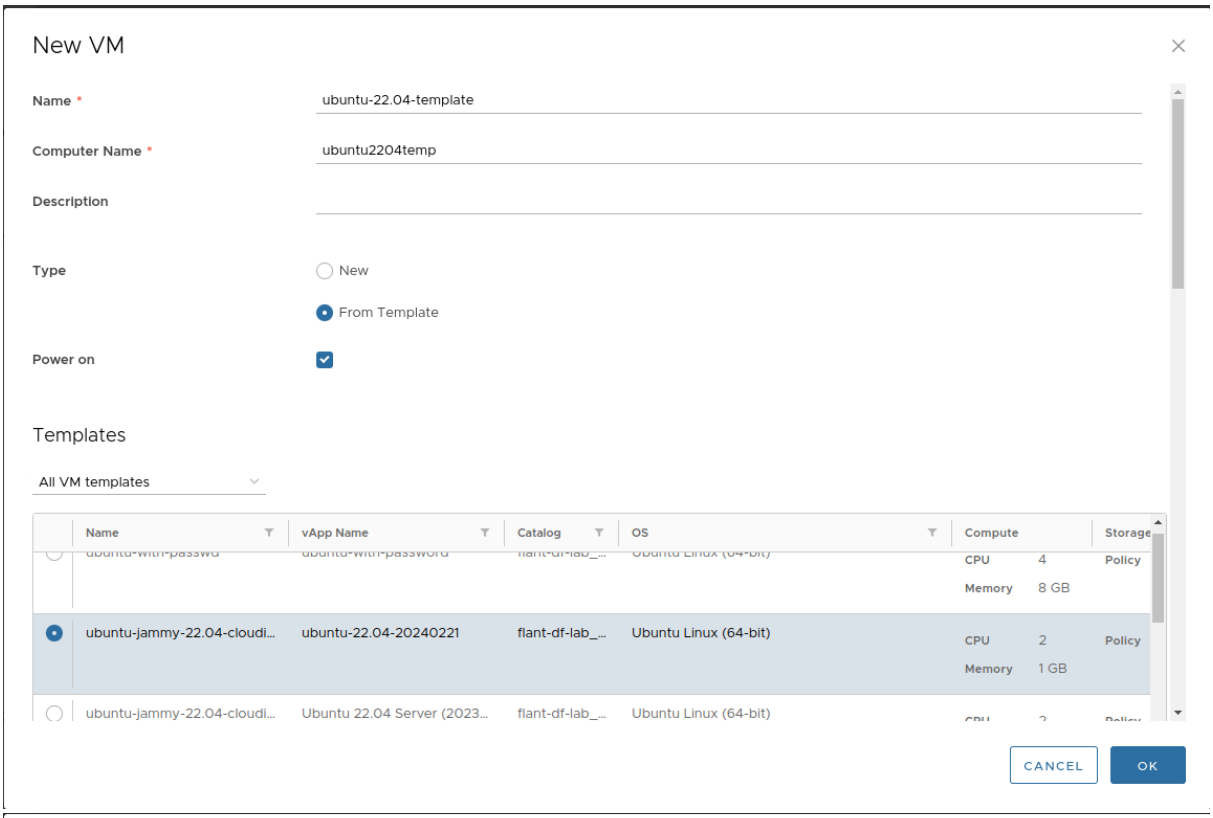
3. Upload the downloaded template to the catalog:

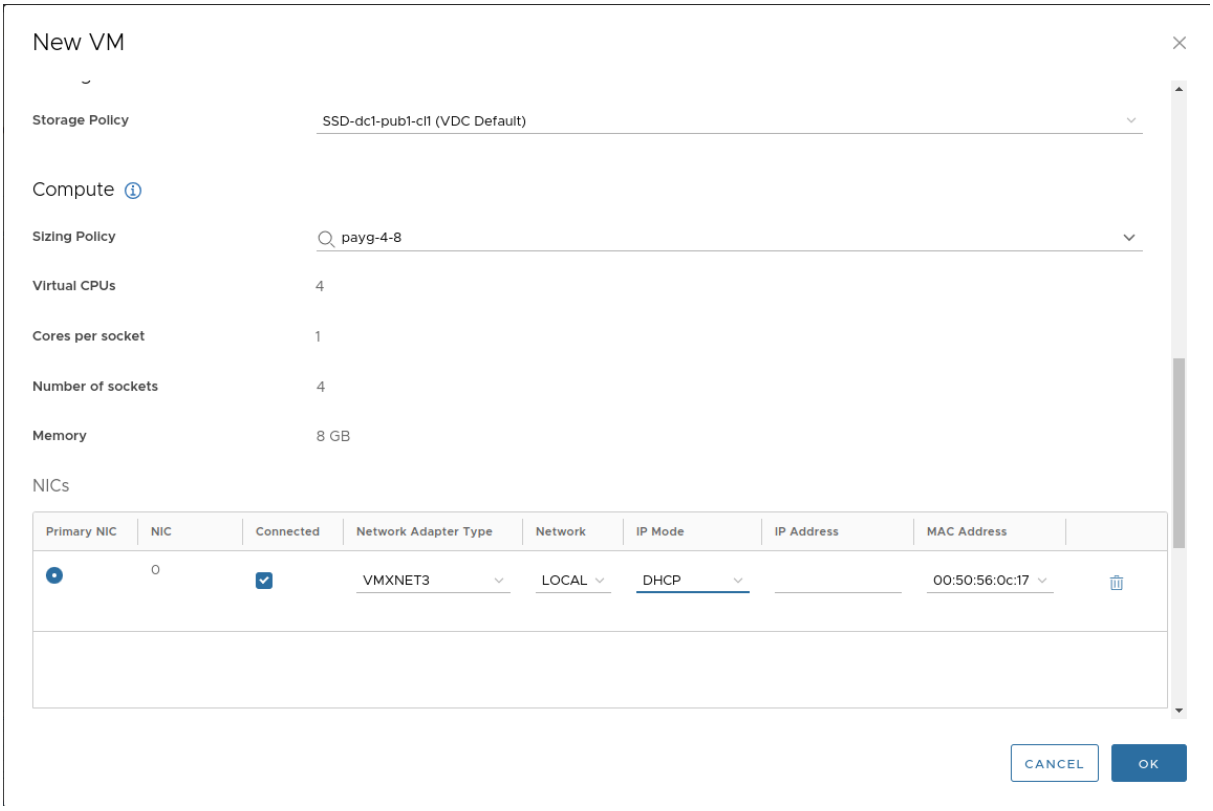




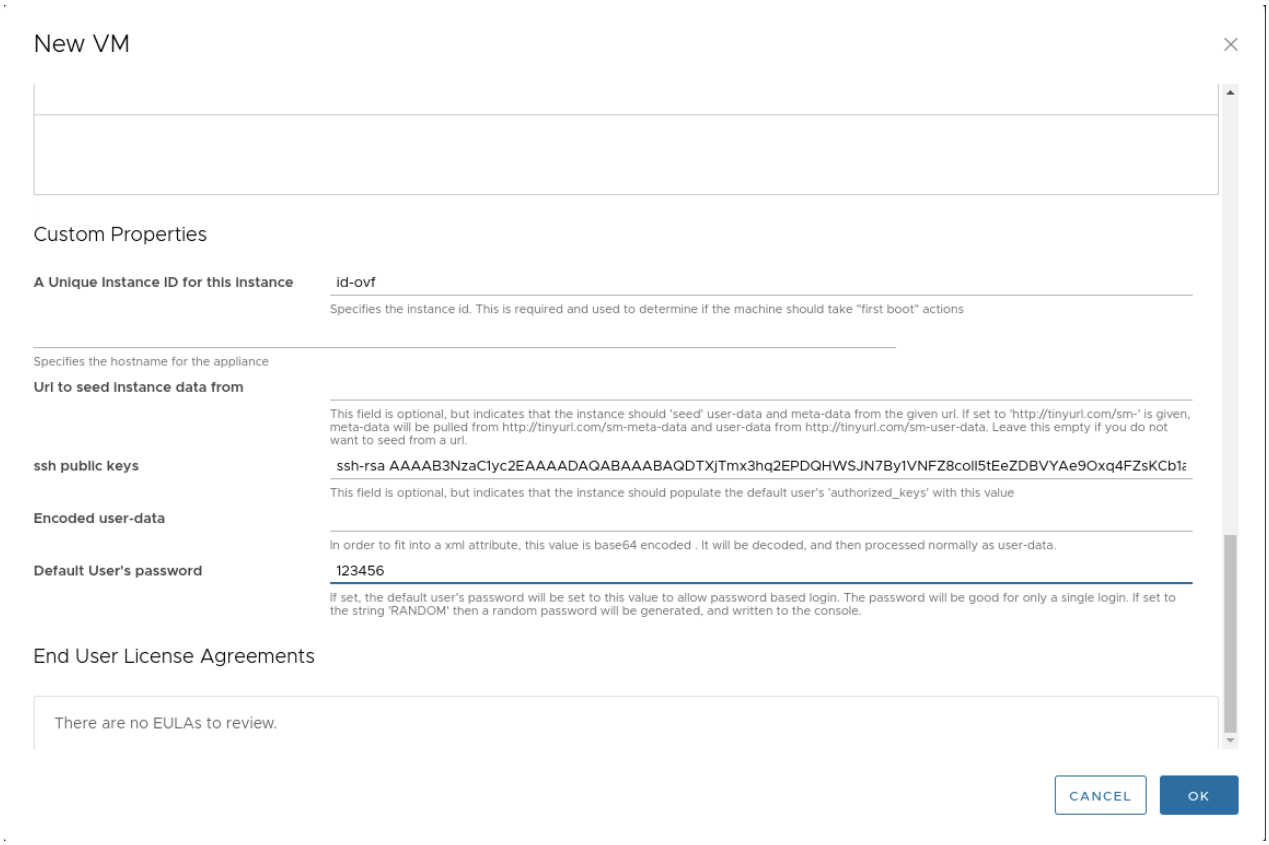


4. Create a VM from the template:



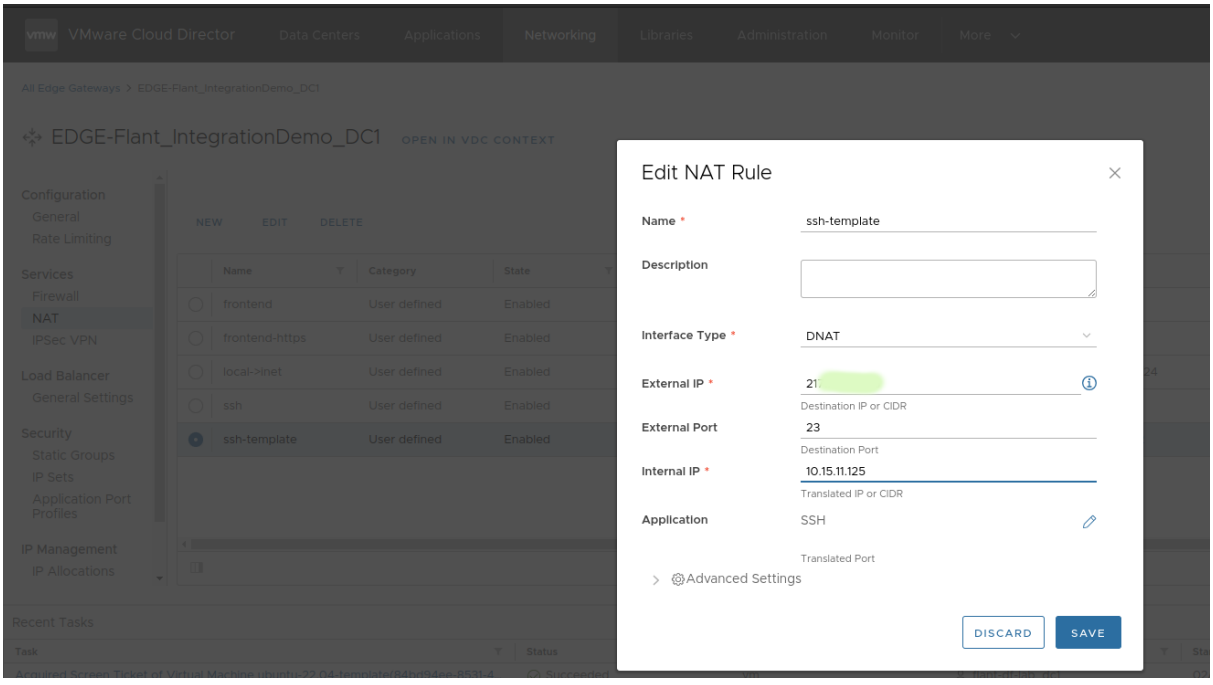


⚠ Set a default password and public key. These will be needed to log into the VM console.



To connect to the VM:

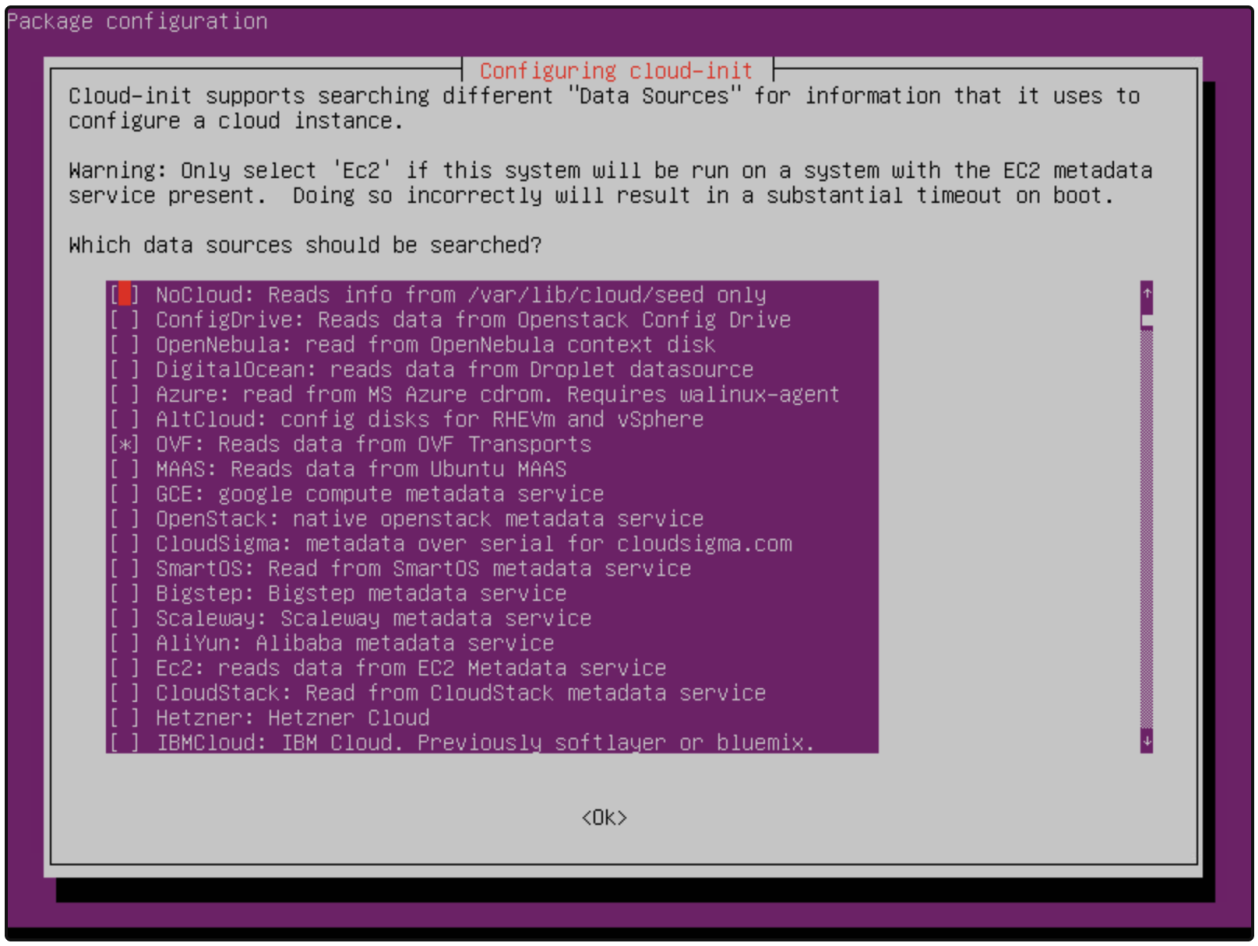
1. Start the VM.
2. Wait until it gets an IP address.
3. Forward port “22” to the VM:



Log into the VM via SSH and run:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91_vmware_cust.cfg
dpkg-reconfigure cloud-init
```

In the dialog window, leave only the checkbox for `OVF: Reads data from OVF transports`. Disable all other options:



Make sure that the `datasource_list` parameter is specified in the `cloud-init` configuration. You can verify this using the following command:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

If the output is empty, execute the following command:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

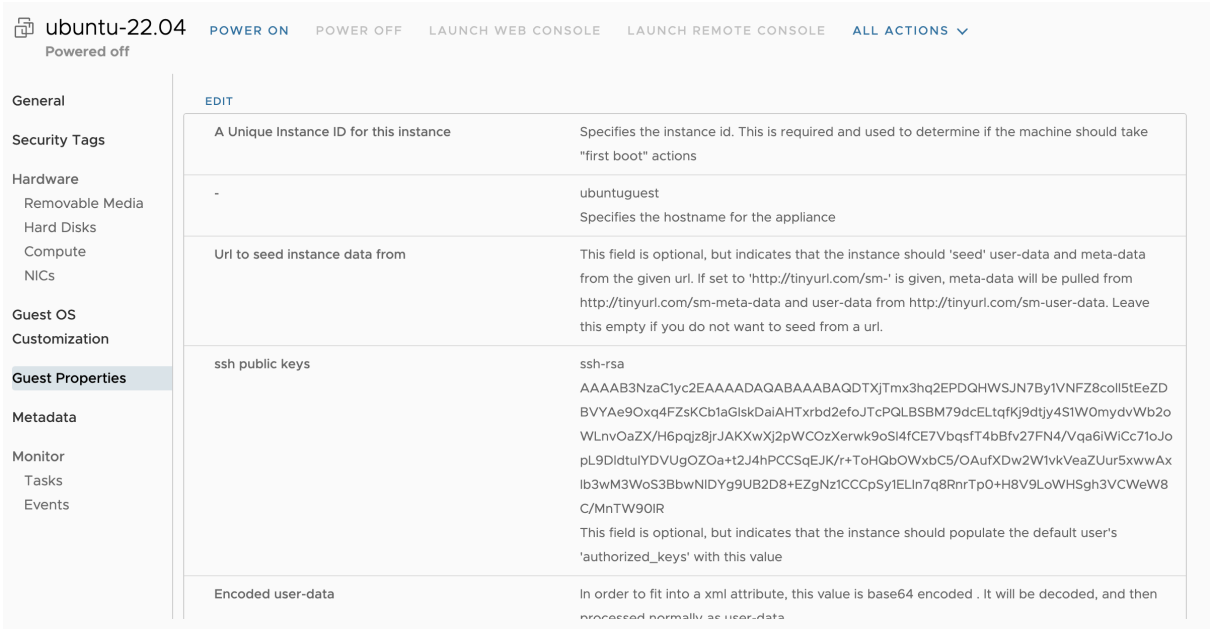
Run the remaining commands:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

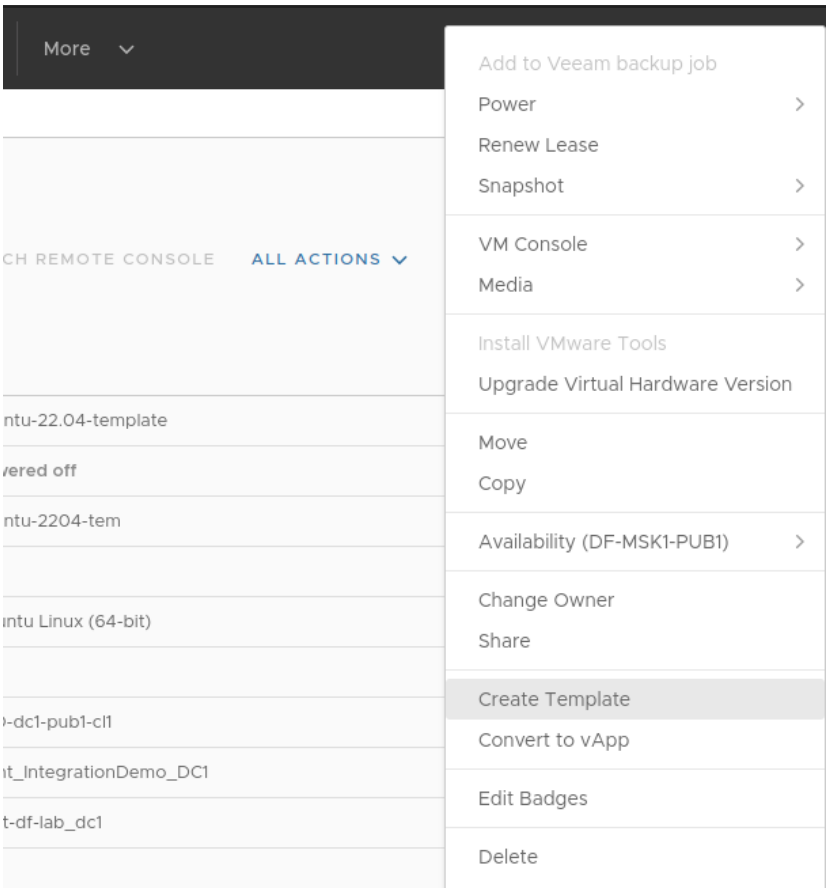
Configuring the template in VCD

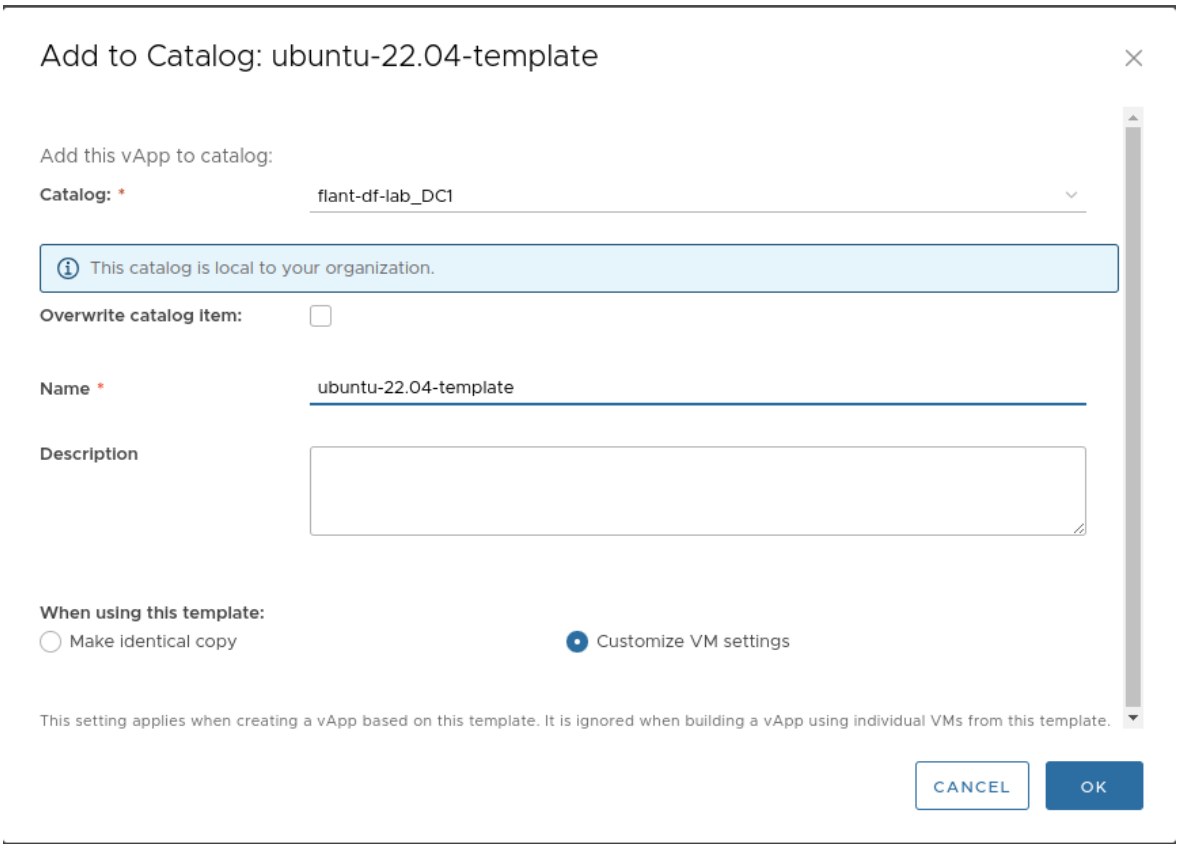
1. Power off the virtual machine and delete all filled “Guest Properties” fields:



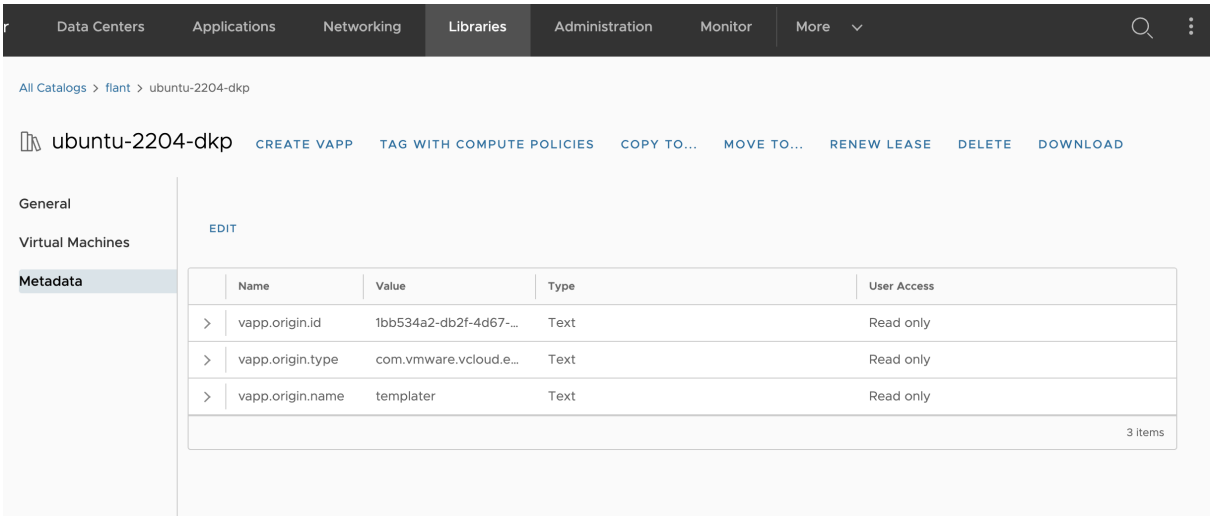


2. Create a virtual machine template:





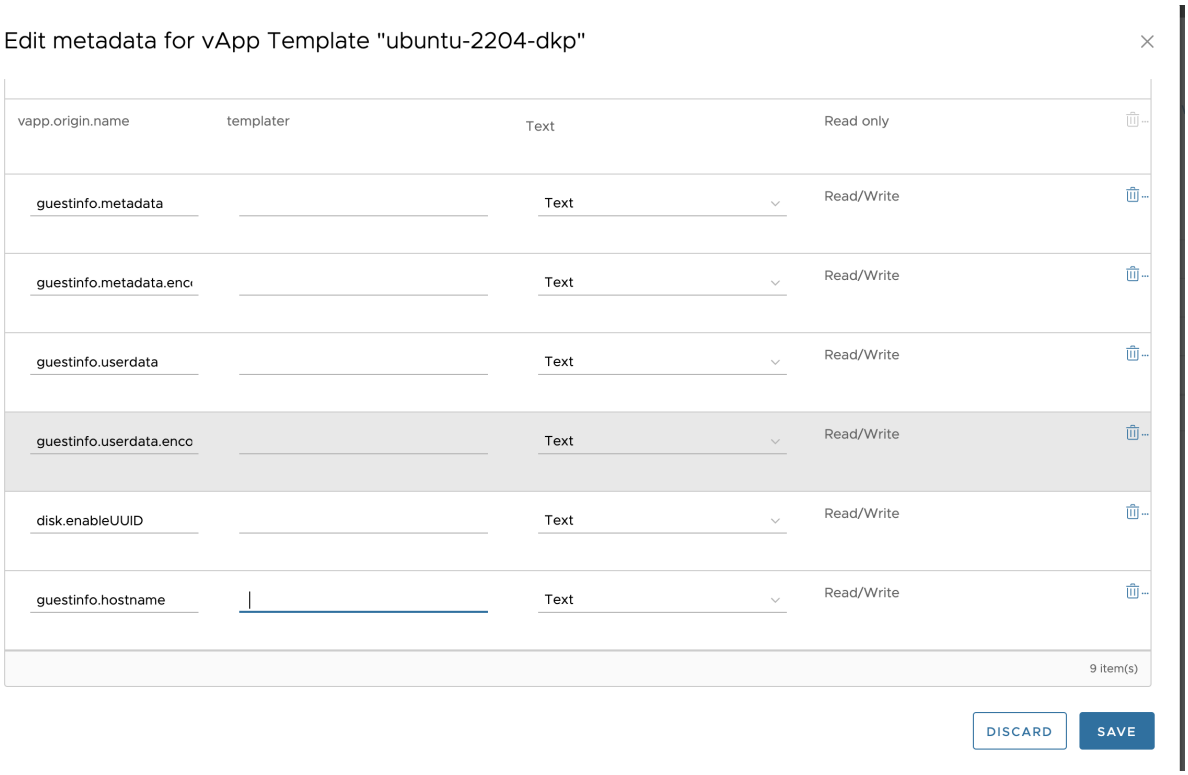
3. In the created template, go to the “Metadata” tab and add six fields:
   - `guestinfo.metadata`
   - `guestinfo.metadata.encoding`
   - `guestinfo.userdata`
   - `guestinfo.userdata.encoding`
   - `disk.enableUUID`
   - `guestinfo.hostname`



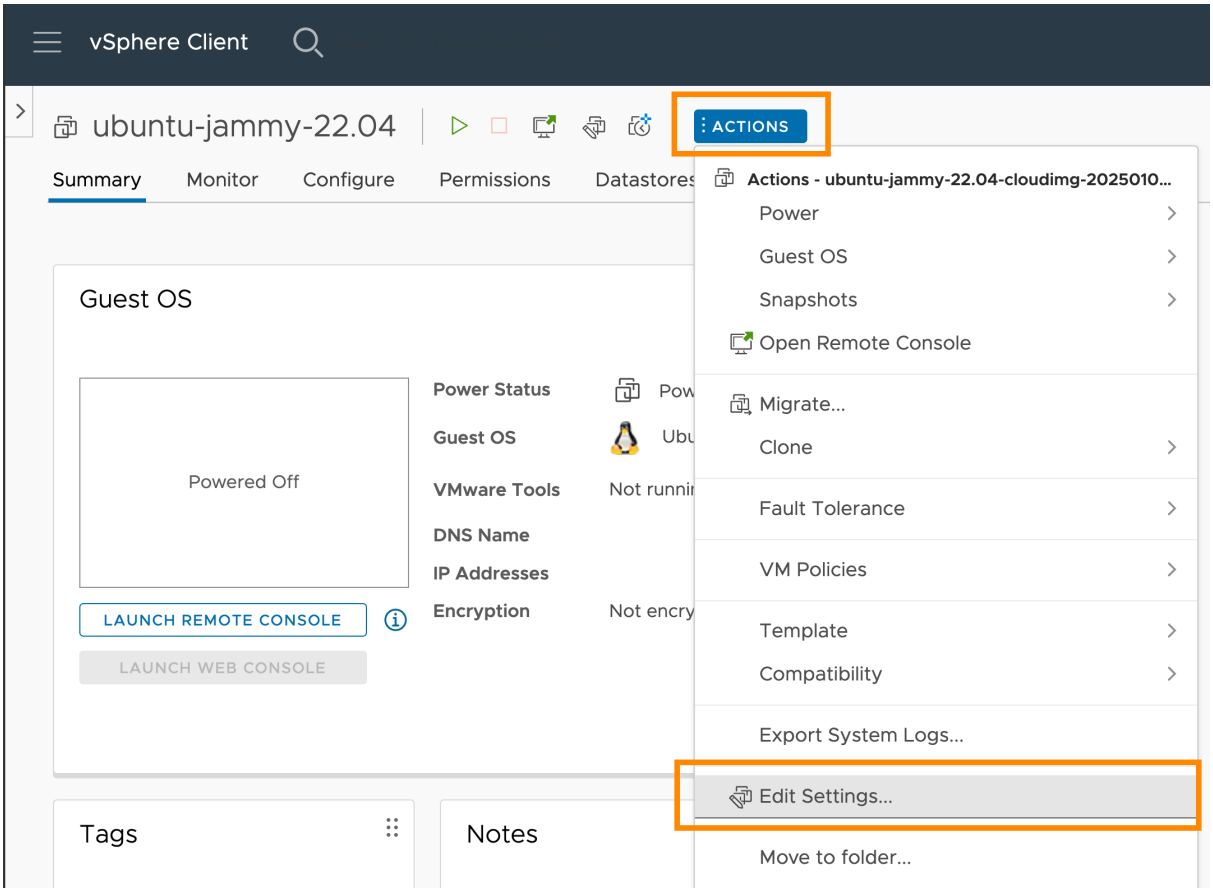
For **each** field in the add/edit form, set the following:

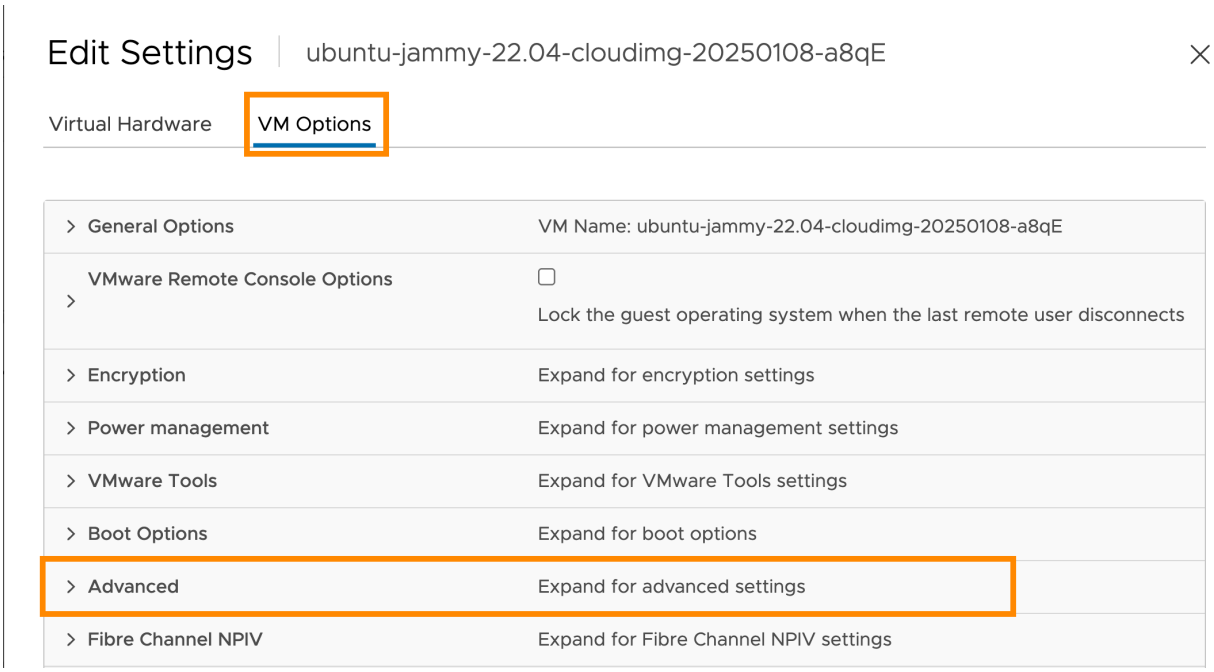
- “Type”: `Text` (text value)
- “User access”: `Read/Write`
- “Value”: A single space

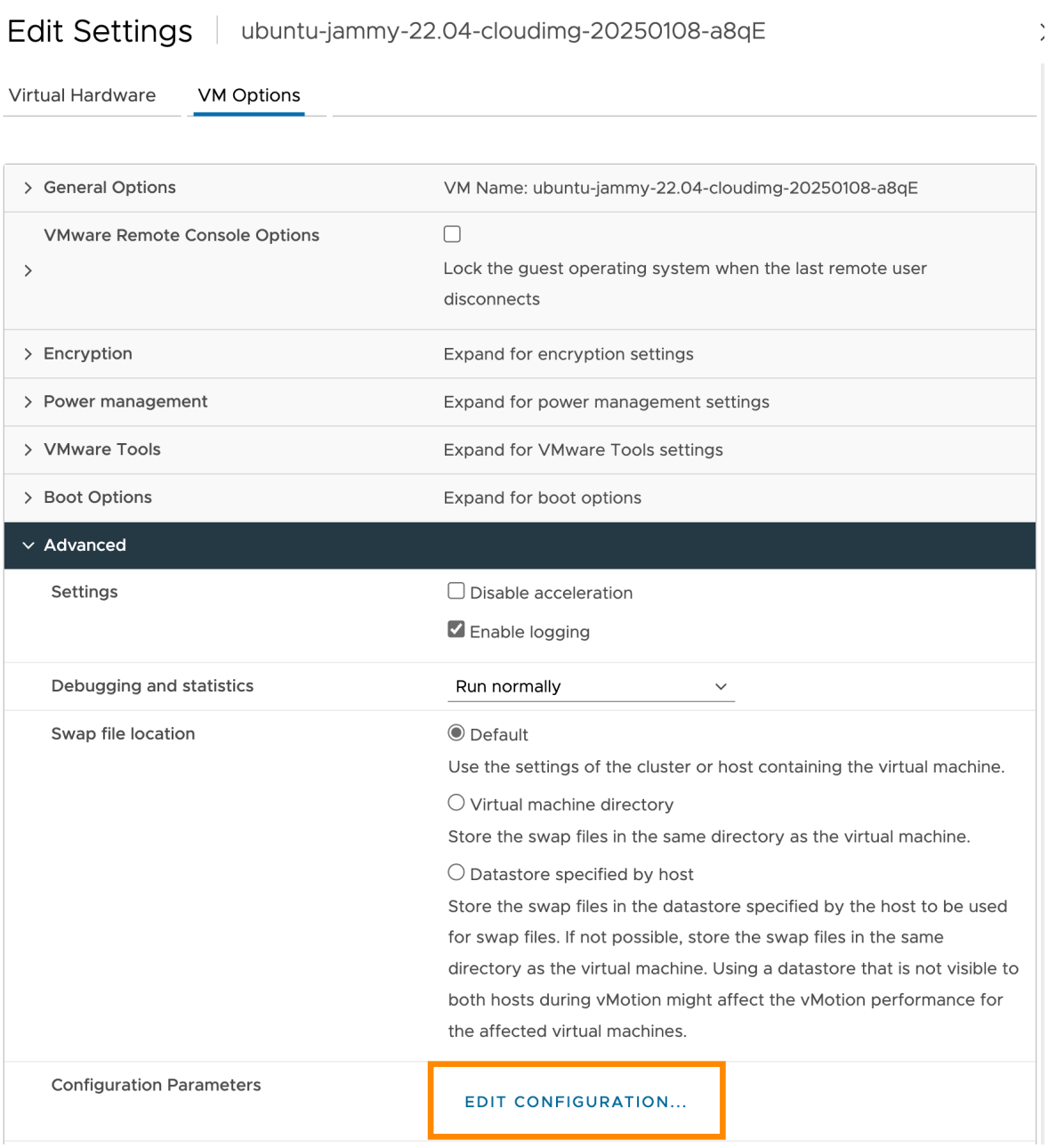
> **i** The VCD UI may not save metadata with an empty value. A single space is used as a placeholder and does not affect functionality. The actual values will be populated automatically when virtual machines are created.

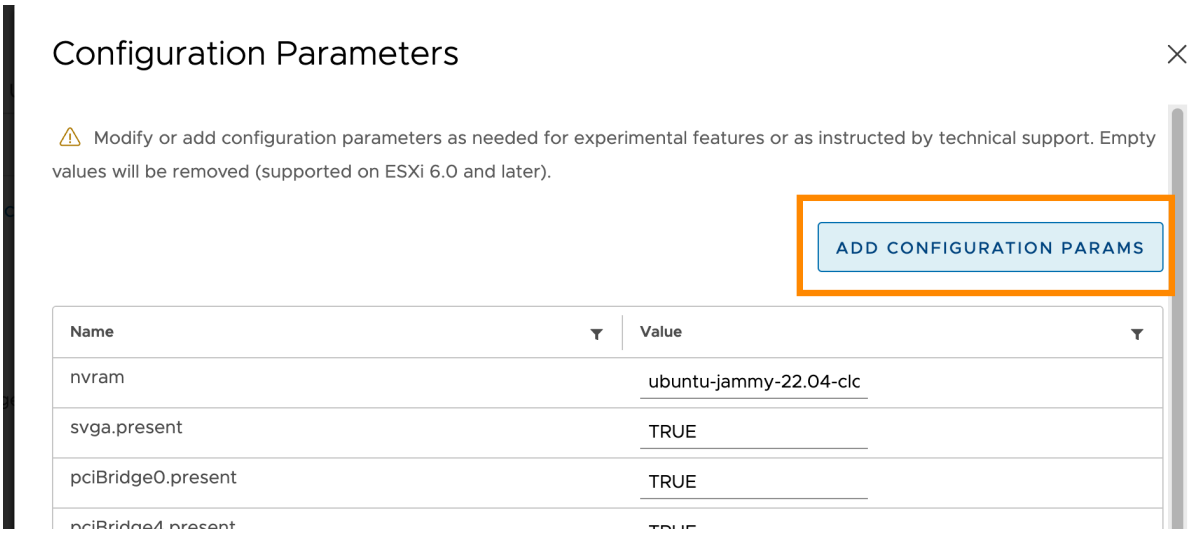


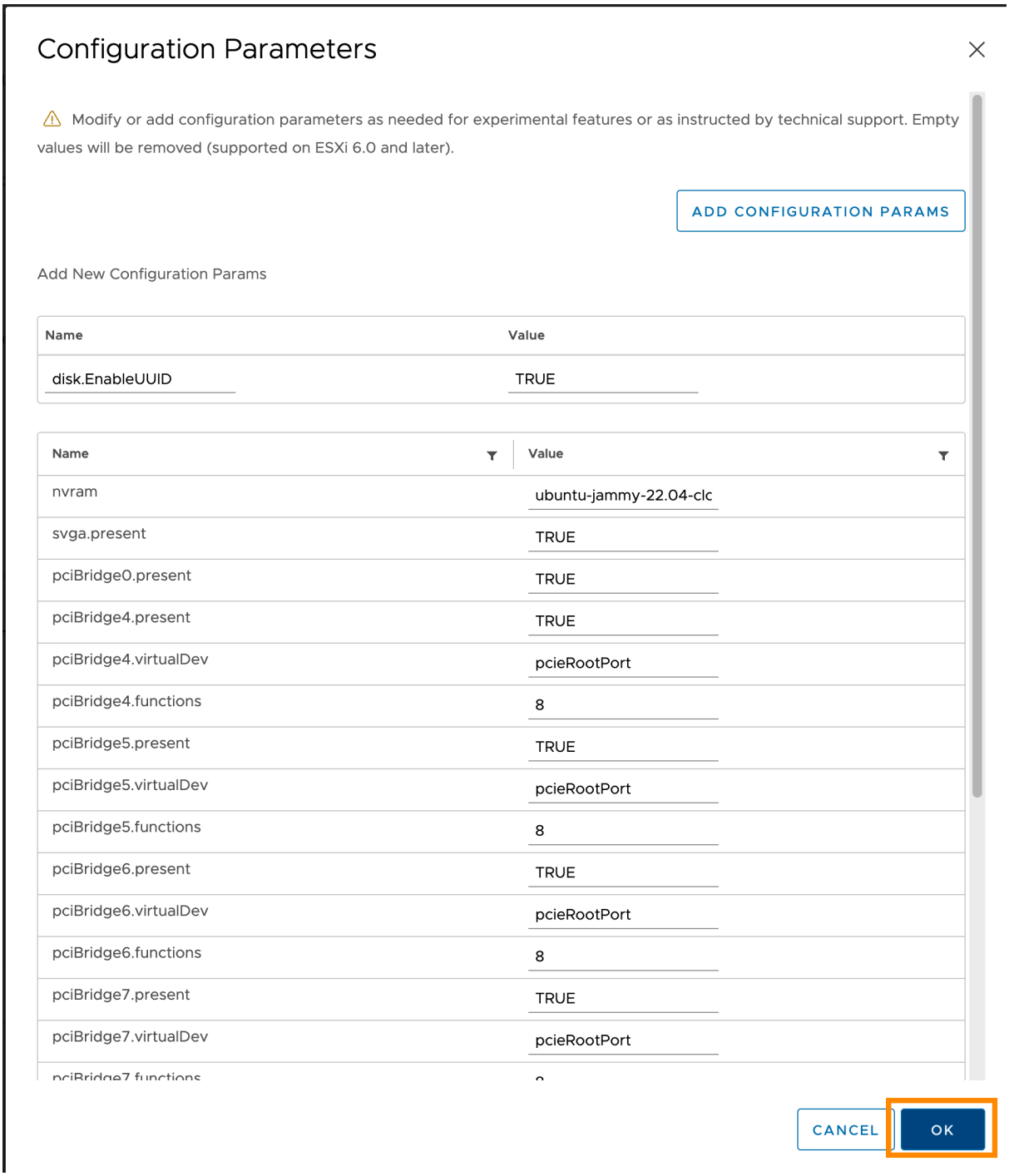
4. In the vCenter management panel, enable the `disk.EnableUUID` parameter for the template:




Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Expand for advanced settings
Fibre Channel NPIV
Expand for Fibre Channel NPIV settings


Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Settings
Disable acceleration
Enable logging
Debugging and statistics
Run normally
Swap file location
Default
Use the settings of the cluster or host containing the virtual machine.
Virtual machine directory
Store the swap files in the same directory as the virtual machine.
Datastore specified by host
Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.
Configuration Parameters
EDIT CONFIGURATION...





Replacing the OS image

Deckhouse Platform creates the nodes from a virtual machine template that is specified in the `template` parameter of the VCDInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the VCDClusterConfiguration

resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new OS image, prepare a new template based on it and follow the steps from “How do I replace the OS image in a cloud cluster?” .

Storage usage

- VCD supports CSI. Disks are created as VCD Independent Disks.
- The guest property `disk.EnableUUID` must be enabled for the VM templates in use.
- Deckhouse Platform supports disk resizing starting from version v1.59.1.

Using the LoadBalancer

- DP components support `Service` resources of type `LoadBalancer` when deployed on VMware Cloud Director (VCD).
- VMware NSX Advanced Load Balancer (ALB or Avi Networks) is used as the load balancer.
- Support is available **only** when using the `NSX-T` network virtualization platform.
- The load balancer functionality must be enabled on the Edge Gateway by your VCD provider. You can verify this under Edge Gateway → Load Balancer → General Settings — the `State` parameter must be `Active`.
- If the load balancer was enabled after the DP cluster was successfully created, the components will automatically pick up the changes within an hour (no additional actions are required).
- For each open port, a Pool + Virtual Service pair is created.
- If a firewall is in place, you must create an allow rule for the load balancer’s external IP address and the corresponding ports.

Additional resources

- Module `cert-manager` documentation
- Module `cloud-provider-vcd` custom resources
- Module `metallb` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in VMware Cloud Director

List of required VMware Cloud Director services

The following VMware Cloud Director service must be available for Deckhouse Platform to operate:

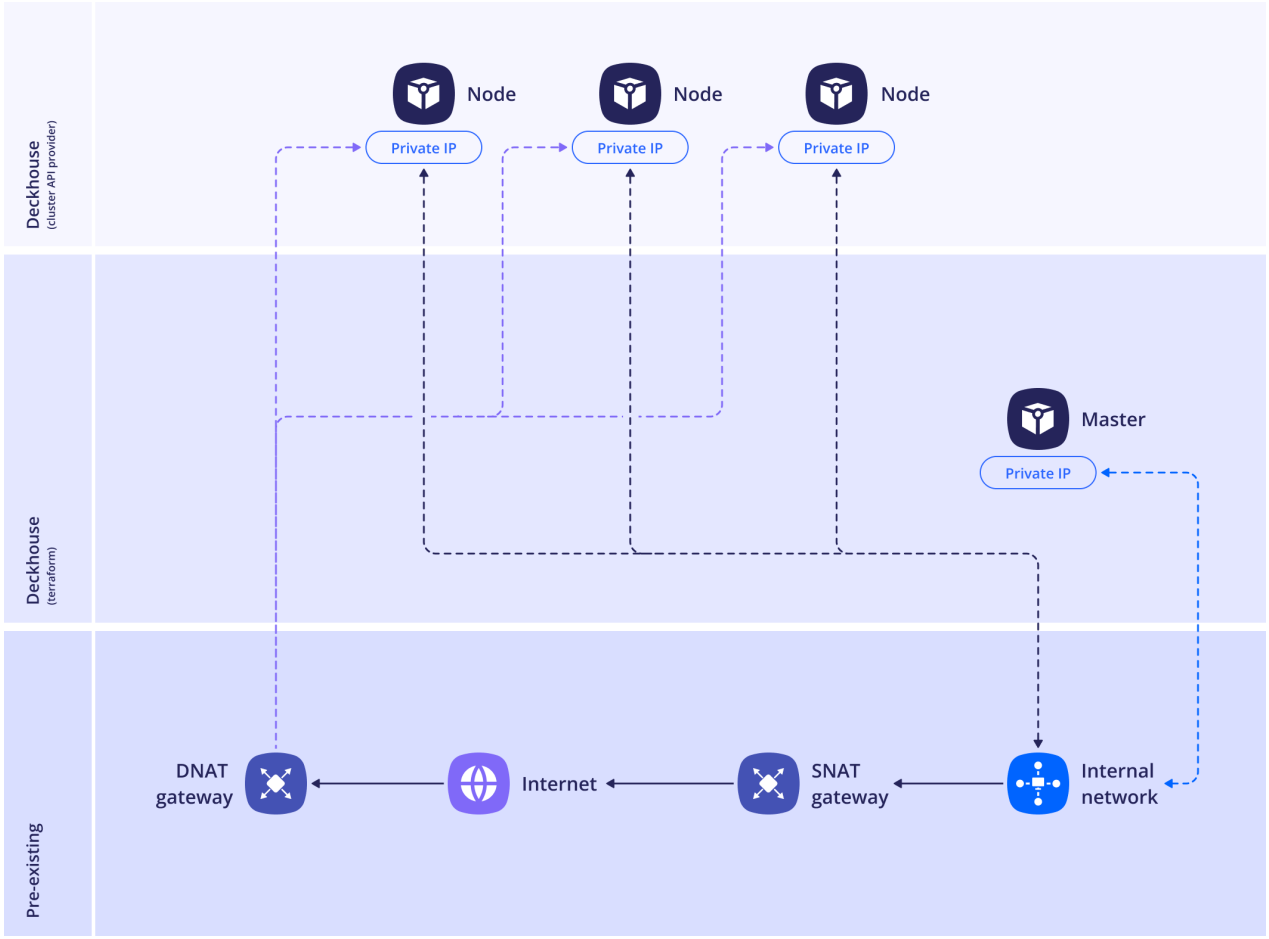
Service	API version
VMware Cloud Director API	`37.2` and later

> For VMware Cloud Director API versions earlier than `37.2`, compatibility mode for legacy API versions is used.

Layouts

Deckhouse Platform supports two layouts for deploying resources in VCD.

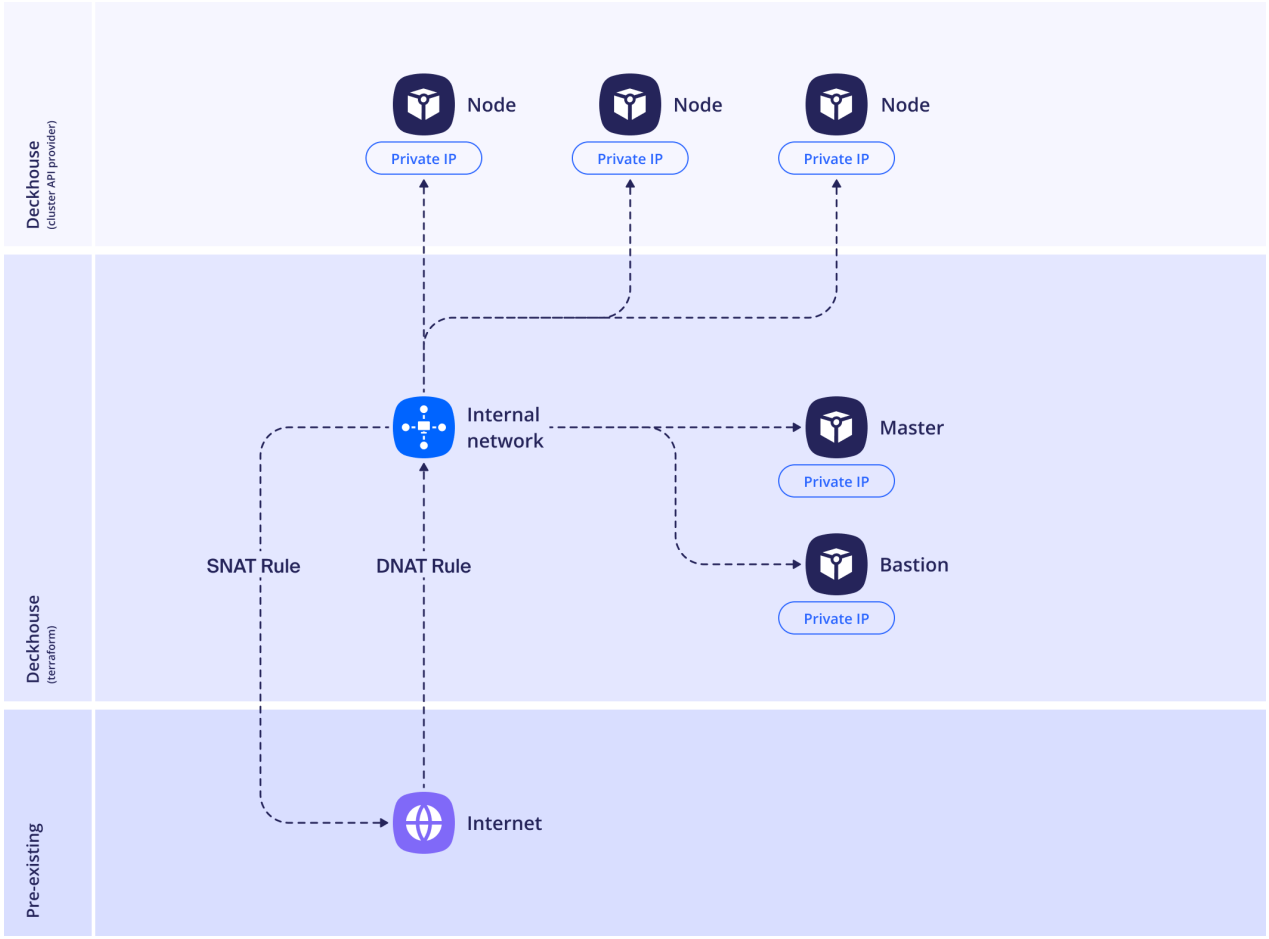
Standard



Example layout configuration:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetwork: internal
    mainNetworkIPAddresses:
    - 192.168.199.2
```

WithNAT



When using this layout, you must confirm the type of network virtualization platform with your administrator and specify it in the `edgeGateway.type` parameter. Two options are supported: `NSX-T` and `NSX-V`.

To ensure administrative access to the cluster nodes, a bastion is deployed. The parameters for its configuration are described in the `bastion` section.

If the Edge Gateway is based on `NSX-T`, a DHCP server will be automatically enabled for the created node network. It will assign IP addresses starting from the 30th address in the subnet up to the penultimate one (before the broadcast address). The starting address of the DHCP pool can be changed using the `internalNetworkDHCPPoolStartAddress` parameter.

If `NSX-V` is used, DHCP must be configured manually. Otherwise, nodes waiting to obtain an IP address via DHCP will not be able to receive one.

⚠ It is not recommended to use dynamic addressing for the first master node together with `NSX-V`.

This layout assumes the automatic creation of the following NAT rules:

- **SNAT** — translation of internal node network addresses to the external address specified in `edgeGateway.externalIP`.
- **DNAT** — translating the external address and port, specified in the `edgeGateway.externalIP` and `edgeGateway.externalPort` properties, respectively, to the internal address of the bastion instance on port 22 using the TCP protocol for administrative access to the nodes via SSH.

⚠ If the Edge Gateway is powered by `NSX-V`, you must specify the name and type of the network to which the rule will be bound in the `edgeGateway.NSX-V.externalNetworkName` and `edgeGateway.NSX-V.externalNetworkType` properties. Usually, this is the network connected to the Edge Gateway in the `Gateway Interfaces` section and having an external IP address.

It is also possible to create firewall rules using the `createDefaultFirewallRules` property.

⚠ If the Edge Gateway is powered by `NSX-T`, existing Edge Gateway rules will be overwritten. This option is intended for scenarios where only one cluster is deployed on the Edge Gateway.

The following rules will be created:

- Allow all outgoing traffic;
- Allow incoming TCP traffic on port 22 for SSH connections to cluster nodes;
- Allow all incoming ICMP traffic;
- Allow incoming TCP and UDP traffic on ports 30000–32767 for `NodePort` services.

Example configuration for the WithNAT layout using `NSX-T`:

```
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Example configuration for the WithNAT layout using `NSX-V`:

```
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Configuration

Integration is carried out using the VCDClusterConfiguration resource, which describes the configuration of the cloud cluster in VCD and is used by the virtualization system if the cluster's control plane is hosted there. The DP module responsible for integration is automatically configured based on the selected deployment layout.

To modify the configuration in a running cluster, execute the following command:

```
d8 -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller edit provider-cluster-configuration
```

> **i** After changing the node parameters, you must run the `dhctl converge` command for the changes to take effect.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
  - name: worker
    replicas: 1
    instanceClass:
      template: Org/Templates/ubuntu-focal-20.04
      sizingPolicy: 16cpu32ram
      storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VCD cloud provider, the instance class is a custom resource called VCDInstanceClass, which contains the specific configuration of the VMs.

The following is the example configuration of VCDInstanceClass for ephemeral nodes of the VMware Cloud Director cloud provider.

Example configuration of VCDInstanceClass

```
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova
```

Storage

A StorageClass is automatically created for each Datastore and DatastoreCluster in the zone (or zones).

You can set the name of StorageClass that will be used in the cluster by default (the `default` parameter), and filter out the unnecessary StorageClasses (the `exclude` parameter).

CSI

By default, the storage subsystem uses CNS volumes capable of on-the-fly resizing. FCD volumes are also supported, but only in the legacy mode. You can set this via the `compatibilityFlag` parameter.

Additional resources

- Module `cloud-provider-vcd` provider configuration
- Module `cloud-provider-vcd` custom resources
- Module `node-manager` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in VMware vSphere

Requirements

The following prerequisites must be met for Deckhouse Platform to work correctly with VMware vSphere:

- Access to vCenter;
- A user account with the required set of privileges;
- Tags and tag categories created in vSphere;
- Networks with DHCP and Internet access;
- Shared Datastore resources available on all ESXi hosts in use;
- vSphere version `7.x` or `8.x` with support for `Online volume expansion`;
- vCenter accessible from inside the cluster from the master nodes;
- A configured Datacenter that includes the following objects:
  1. VirtualMachine template:
     - The virtual machine image must use `Virtual machines with hardware version 15 or later` — this is required for online resize support.
     - The image must include the `open-vm-tools`, `cloud-init`, and `cloud-init-vmware-guestinfo` packages if the `cloud-init` version is lower than `21.3`.
  2. Network:
     - The network must be available on all ESXi hosts where virtual machines are planned to be created.
  3. Datastore (one or more):
     - The Datastore must be connected to all ESXi hosts where virtual machines are planned to be created.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Datastore (default: `k8s-zone`). This tag defines the zone.
     - All Cluster objects within the same zone must have access to all Datastore objects in that zone.
  4. Cluster:
     - All ESXi hosts in use must be added to the Cluster.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Cluster (default: `k8s-zone`). This tag defines the zone.
     - DRS must be enabled in the Cluster. The platform does not pick a specific host when creating a virtual machine, so vSphere chooses the host.
  5. Folder for the virtual machines being created:
     - This parameter is optional.
     - By default, the root virtual machine folder is used.
  6. Role:
     - The role must include the required set of privileges.
  7. User:
     - The user must be assigned the role specified in the previous item.
- A tag from the category specified in the `regionTagCategory` parameter must be assigned to the created Datacenter (default: `k8s-region`). This tag defines the region.

VM image requirements

To create a VM template (Template), it is recommended to use a ready-made cloud image/OVA file provided by the OS vendor:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (section *Generic Cloud / OpenStack*)

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Preparing the virtual machine image

⚠ Disable VMware Guest OS Customization (and any vApp/OS customization mechanisms, if applicable in your setup) for the template and the cluster virtual machines. DP performs the initial node configuration via `cloud-init` (VMware GuestInfo datasource). Enabled customization can conflict with `cloud-init` and lead to incorrect node initialization.

1. Install the required packages:

   If you use `cloud-init` version lower than 21.3 (VMware GuestInfo support is required):

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
   ```

   If you use `cloud-init` version 21.3 or higher:

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init
   ```

2. Verify that the `disable_vmware_customization: false` parameter is set in `/etc/cloud/cloud.cfg`.
3. Make sure the `default_user` parameter is specified in `/etc/cloud/cloud.cfg`. It is required to add an SSH key when the VM starts.

4. Add the VMware GuestInfo datasource — create `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

```
datasource:
  VMware:
    vmware_cust_file_max_wait: 10
```

5. Before creating the VM template, reset the `cloud-init` identifiers and state using the following commands:

```
truncate -s 0 /etc/machine-id &&
rm /var/lib/dbus/machine-id &&
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

6. Clear `cloud-init` event logs:

```
cloud-init clean --logs --seed
```

⚠ After the virtual machine starts, the following services related to the packages installed during `cloud-init` preparation must be running:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

To ensure that the services are enabled, use the command:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Example output for enabled services:

```
enabled
enabled
enabled
```

i DP creates VM disks of type `eagerZeroedThick`, but the type of disks of created VMs may be changed without notification according to the `VM Storage Policy` settings in vSphere.
For more details, see the documentation.

i DP uses the `ens192` interface as the default interface for VMs in vSphere. Therefore, when using static IP addresses in `mainNetwork`, you must create an interface named `ens192` in the OS image as the default interface.

Replacing the OS image

Deckhouse Platform creates the nodes from a virtual machine template that is specified in the `template` parameter of the VsphereInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the VsphereClusterConfiguration resource for `CloudPermanent` nodes and master nodes. To move the cluster nodes to a new OS image, prepare a new template based on it and follow the steps from “How do I replace the OS image in a cloud cluster?”.

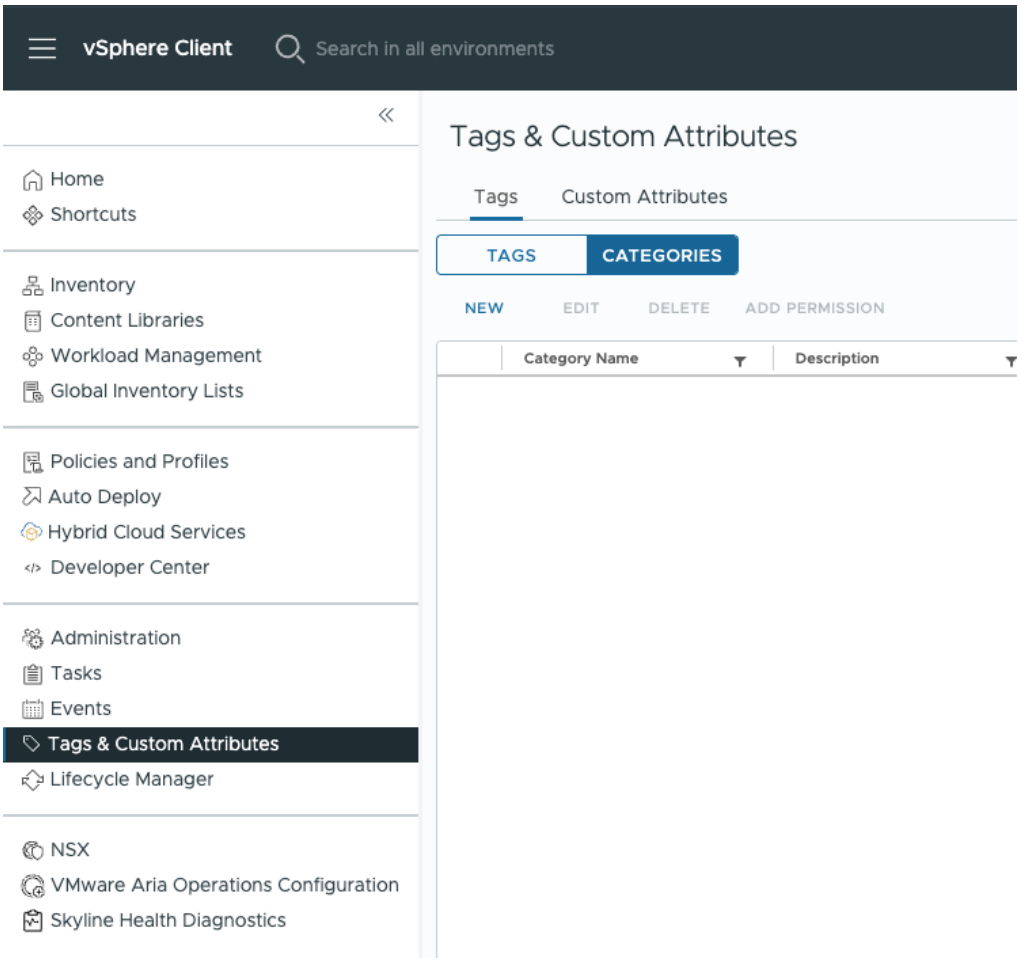
vSphere configuration

Configuration in vSphere Client

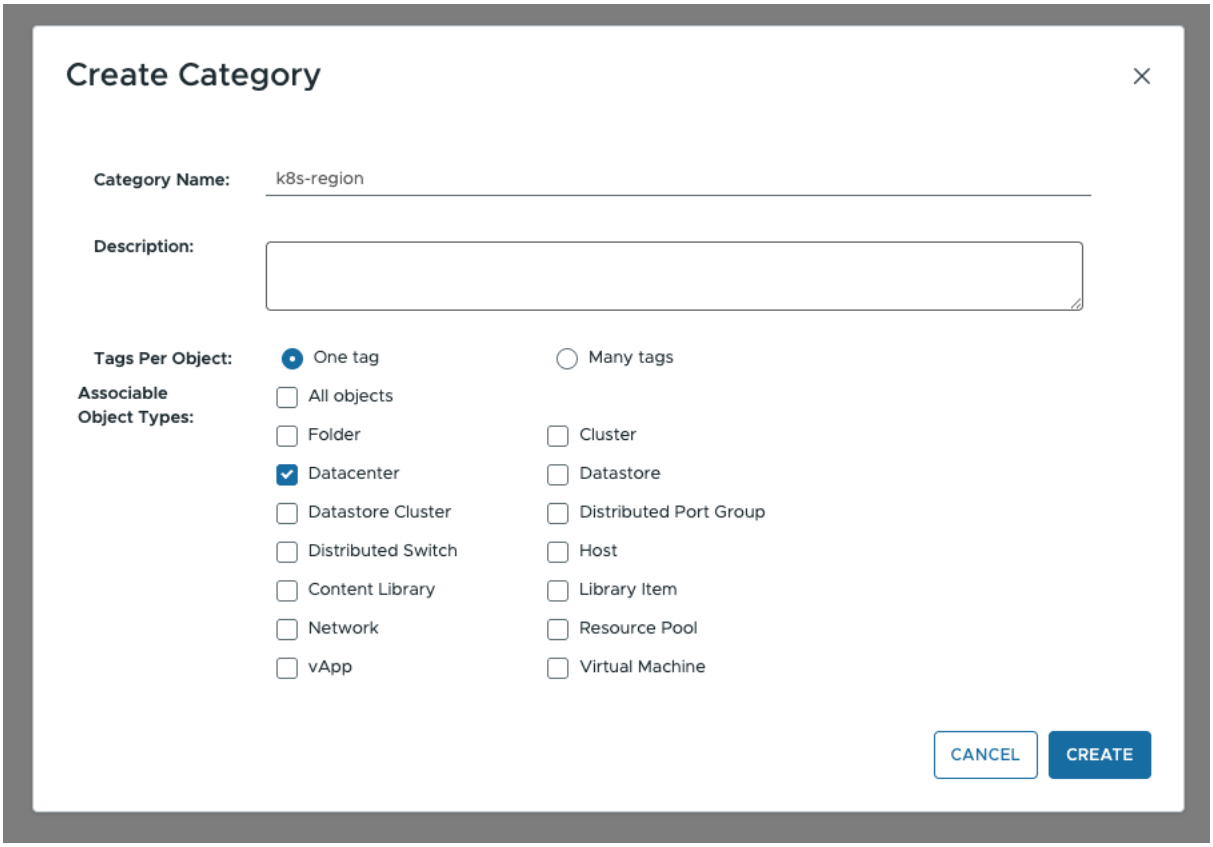
Creating tags and tag categories in vSphere Client

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

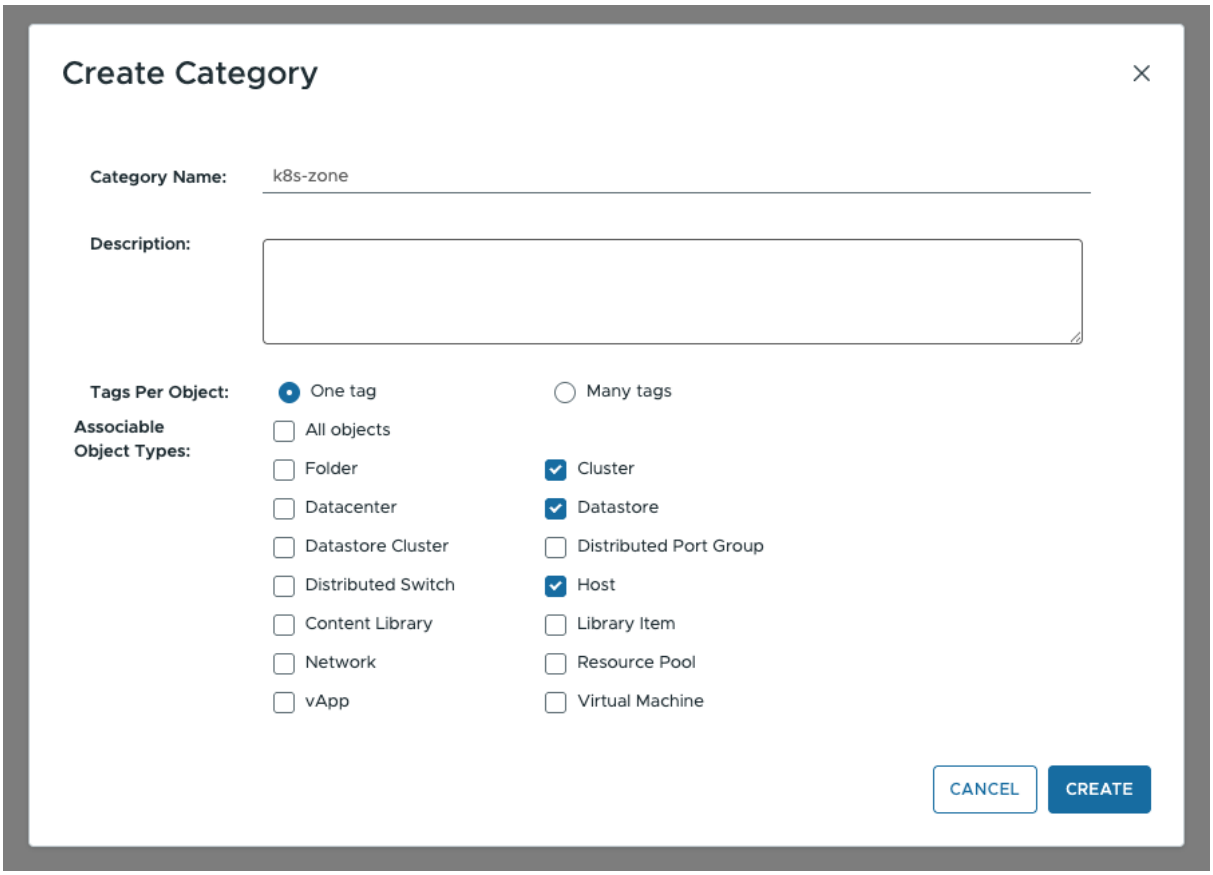
1. Open vSphere Client and go to “Menu” → “Tags & Custom Attributes” → “Tags”.



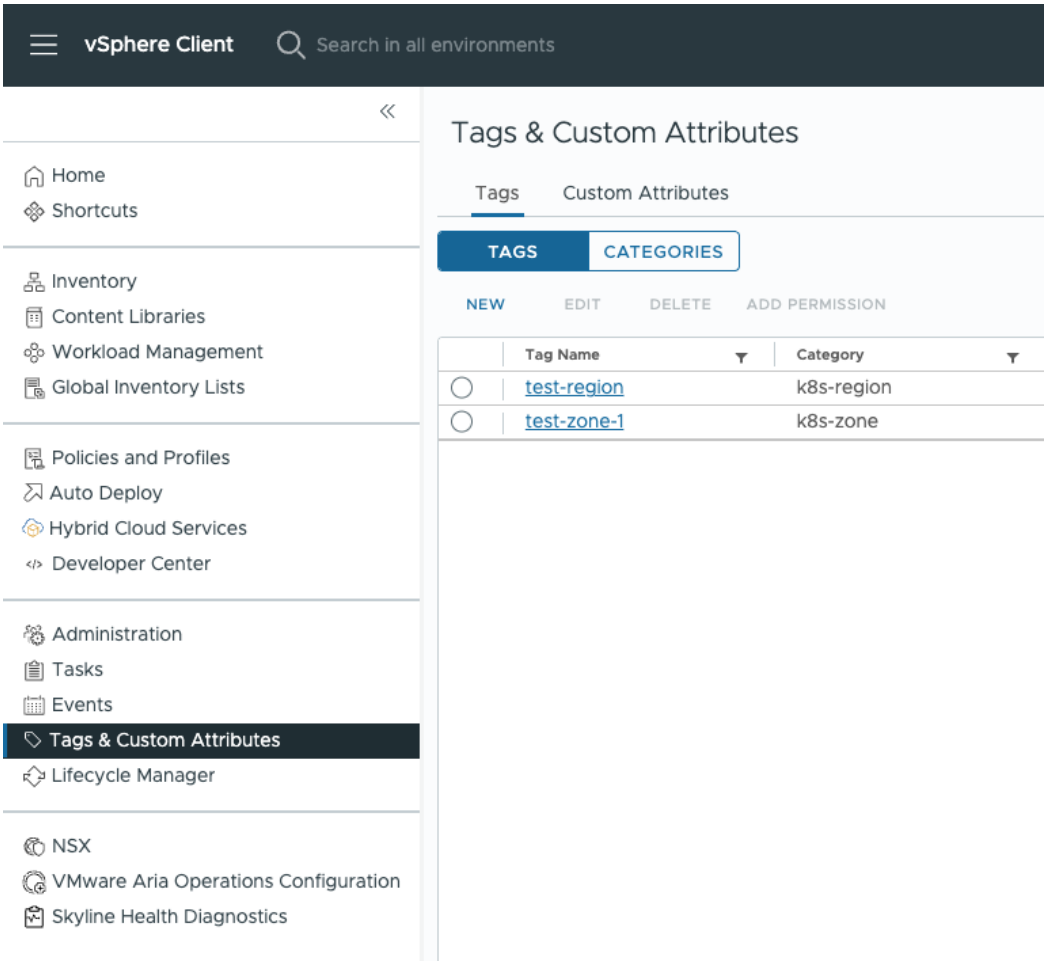
2. Open the “Categories” tab and click “NEW”. Create a category for regions (for example, `k8s-region`): set “Tags Per Object” to “One tag” and specify the applicable object types, including Datacenter.



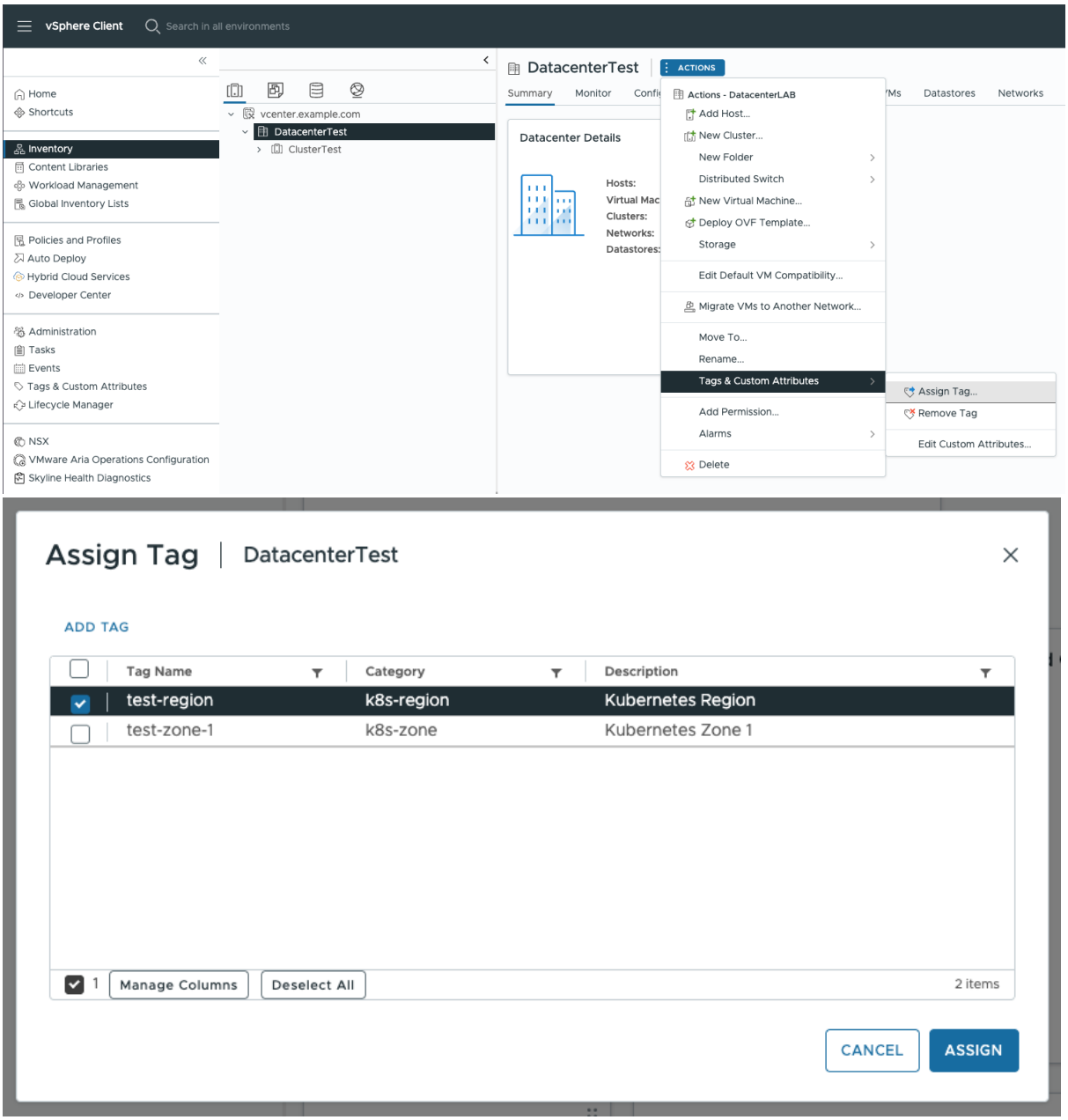
3. Create a second category for zones (for example, `k8s-zone`) with the object types Host, Cluster, and Datastore.



4. Go to the “Tags” tab and create at least one tag in the region category and one tag in the zone category (for example, `test-region`, `test-zone-1`).



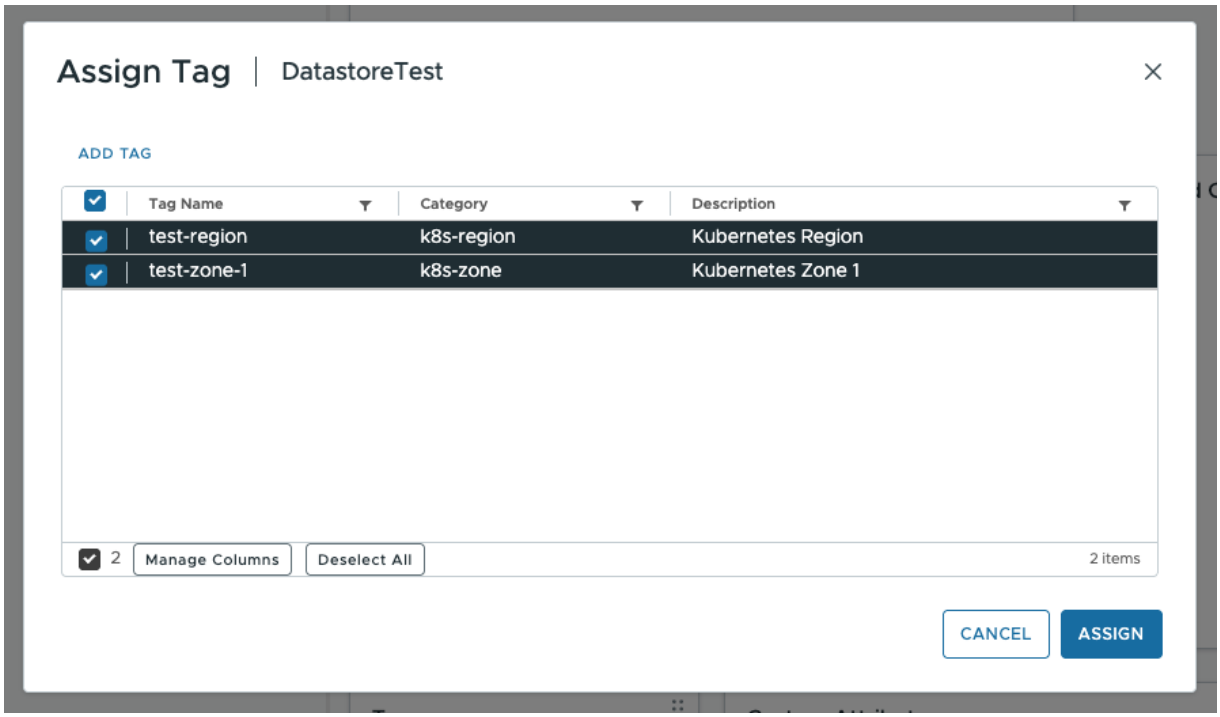
5. In the “Inventory” tab, select the target Datacenter, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag” and assign the region tag. Repeat this step for each Cluster that will host nodes, assigning the appropriate zone tags.



Configuring Datastore in vSphere Client

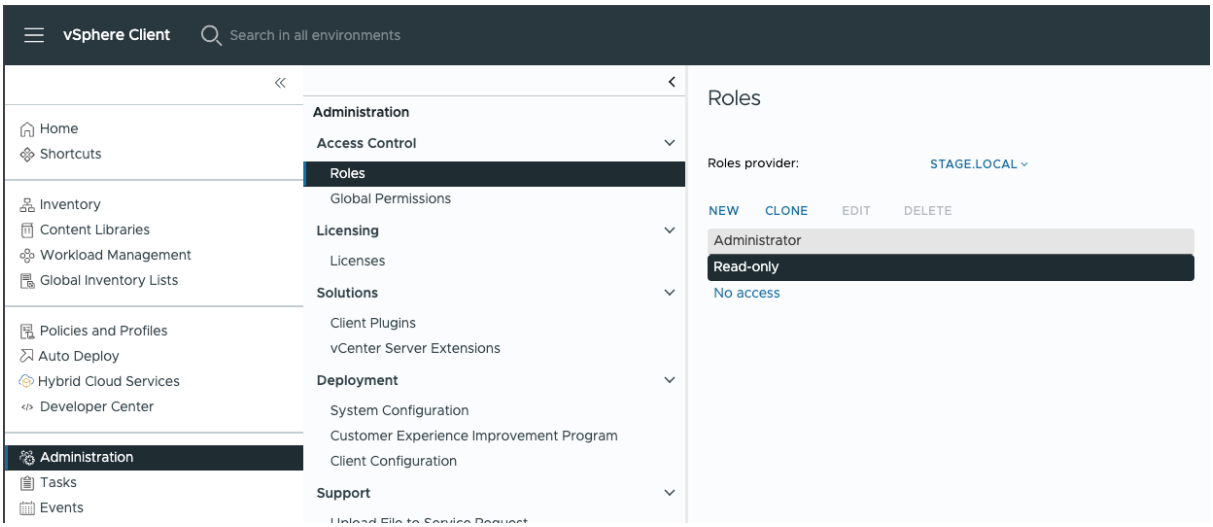
> ⚠ For dynamic provisioning of PersistentVolume, the Datastore must be available on **every** ESXi host in the zone (shared datastore).

In the “Inventory” tab, select the Datastore, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag”. Assign the Datastore the same region tag as the corresponding Datacenter, and the same zone tag as the corresponding Cluster.

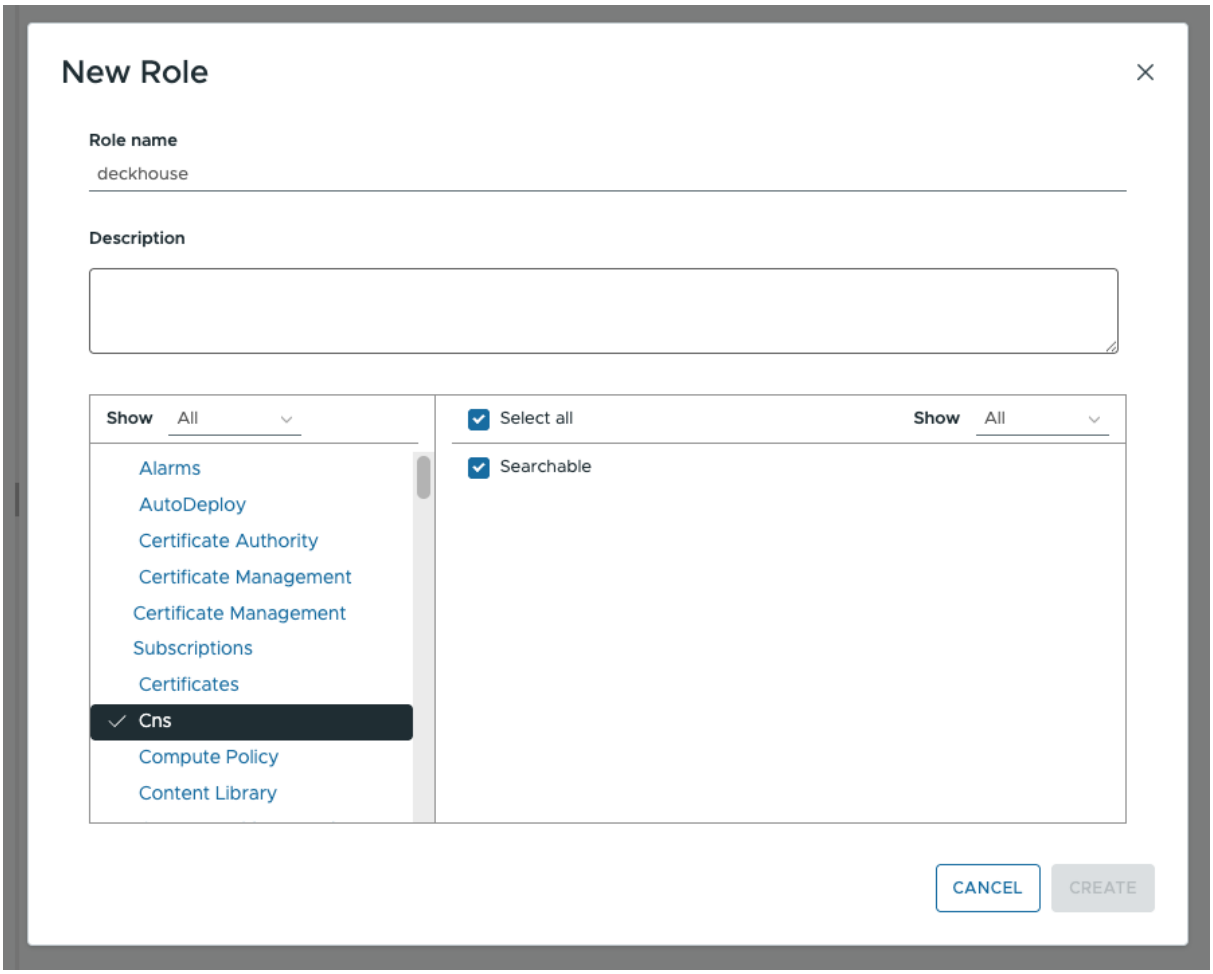


Creating and assigning a role in vSphere Client

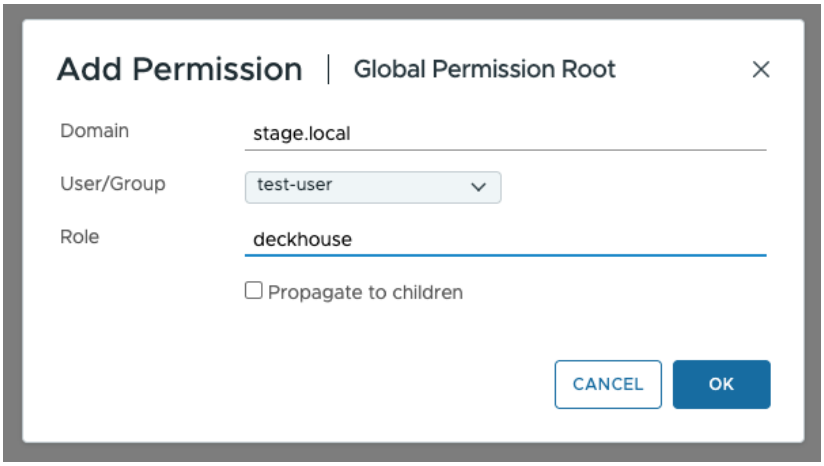
1. Go to “Menu” → “Administration” → “Access Control” → “Roles”.



2. Click “NEW”, enter a role name (for example, `deckhouse`), and add the privileges from the list.



3. Assign the role to the Deckhouse service account: go to “Menu” → “Administration” → “Access Control” → “Global Permissions”, click “ADD”, and select the user and the `deckhouse` role.



Configuration with govc

Installing govc

To continue configuring vSphere, install the govc CLI utility.

After installation, set the environment variables required to connect to `vCenter`.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Creating tags and tag categories with govc

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and

a Cluster acts as a zone. Tags are used to establish this mapping.

Create tag categories with the following commands:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Create tags in each category. If you intend to use multiple “zones” (Cluster), create a tag for each one of them:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Attach the “region” tag to Datacenter:

```
govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>
```

Attach “zone” tags to the Cluster objects:

```
govc tags.attach -C k8s-zone test-zone-1 /<DATACENTER_NAME>/host/<CLUSTER_NAME_1>
govc tags.attach -C k8s-zone test-zone-2 /<DATACENTER_NAME>/host/<CLUSTER_NAME_2>
```

Datastore configuration with govc

⚠ For dynamic PersistentVolume provisioning, a Datastore must be available on **each** ESXi host (shared datastore).

Assign the “region” and “zone” tags to the Datastore objects to automatically create a StorageClass in the Kubernetes cluster:

```
govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_1>
govc tags.attach -C k8s-zone test-zone-1 /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_1>

govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_2>
govc tags.attach -C k8s-zone test-zone-2 /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_2>
```

Creating and assigning a role with govc

i We’ve intentionally skipped User creation since there are many ways to authenticate a user in the vSphere.

The role described below includes the privileges from the list of required privileges section. If you need a more granular Role, please contact your Deckhouse support.

Create a role with the required privileges:

```
govc role.create deckhouse \
    Cns.Searchable \
    Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
    Folder.Create Folder.Delete Folder.Move Folder.Rename \
    Global.GlobalTag Global.SystemTag \
    InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
    InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
    InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
    InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
    InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
    Network.Assign \
    Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
    StorageProfile.View \
    System.Anonymous System.Read System.View \
    VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
    VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
    VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
    VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
    VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
    VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
    VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
    VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
    VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
    VirtualMachine.Config.UpgradeVirtualHardware \
    VirtualMachine.GuestOperations.Query \
    VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
    VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
    VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
    VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
    VirtualMachine.Inventory.Move \
    VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
    VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs
\
    VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Assign the role to a user on the vCenter object.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

i For more detailed permission configuration, refer to the official documentation.

Additional resources

- Module `cloud-provider-vsphere` documentation

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

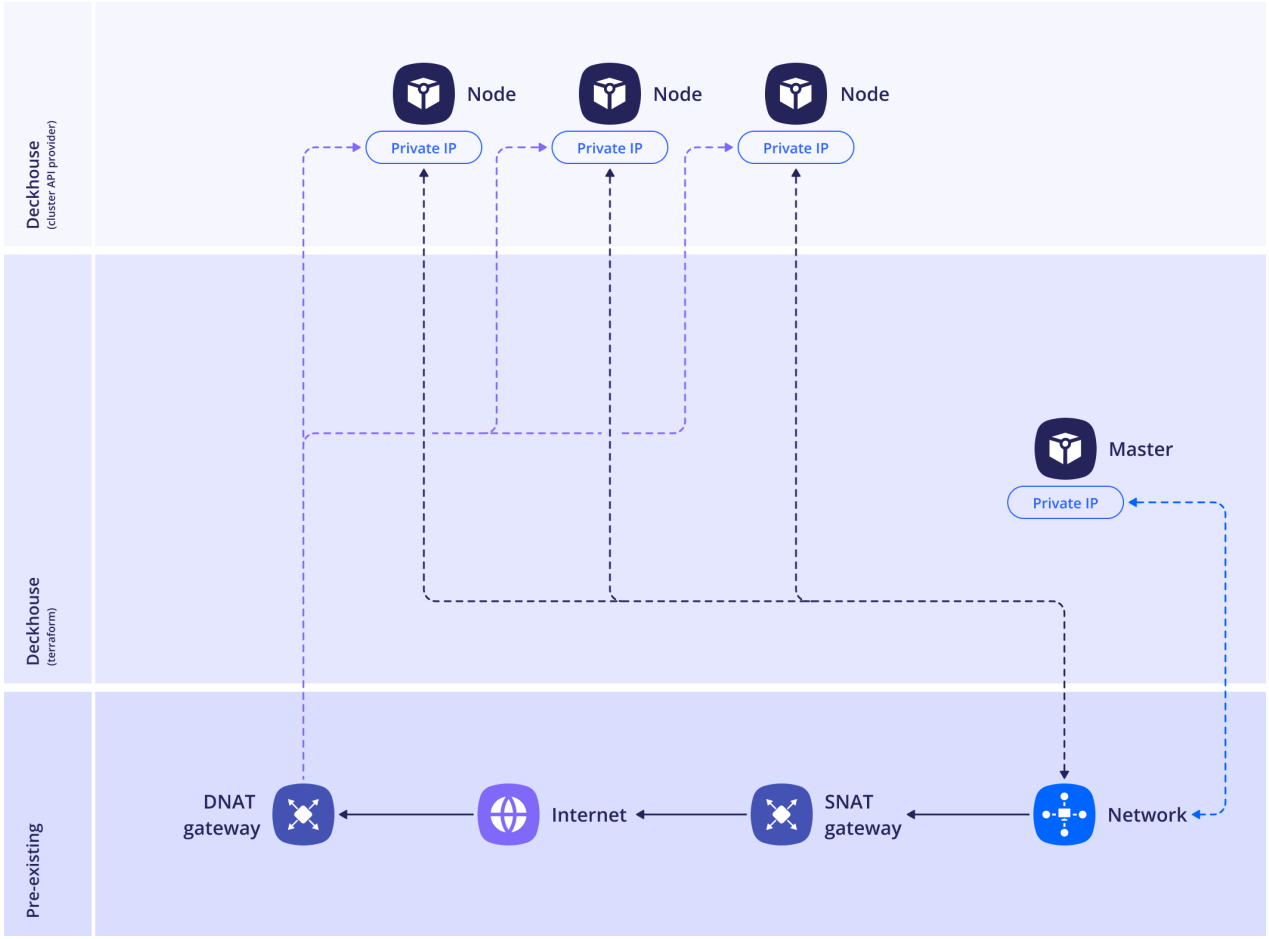
Layouts and configuration in VMware vSphere

Standard

The Standard layout is intended for deploying a cluster within the vSphere infrastructure with full control over resources, networking, and storage.

Key features:

- Uses a vSphere Datacenter as a `region`.
- Uses a vSphere Cluster as a `zone`.
- Supports multiple zones and node placements across zones.
- Supports using different datastores for disks and volumes.
- Supports network connectivity including additional network isolation (for example, MetalLB + BGP).



Example configuration:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
    - ru-central1-a
    - ru-central1-b
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
nodeGroups:
  - name: khm
    replicas: 1
    zones:
      - ru-central1-a
    instanceClass:
      numCPUs: 4
      memory: 8192
      template: dev/golden_image
      datastore: dev/lun_1
      mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
  - ru-central1-a
  - ru-central1-b
```

Required parameters for the VsphereClusterConfiguration resource:

- `region`: Tag assigned to the Datacenter object.
- `zoneTagCategory` and `regionTagCategory`: Tag categories used to identify regions and zones.
- `internalNetworkCIDR`: Subnet for assigning internal IP addresses.
- `vmFolderPath`: Path to the folder where cluster virtual machines will be placed.
- `sshPublicKey`: Public SSH key used to access the nodes.
- `zones`: List of zones available for node placement.

i All nodes placed in different zones must have access to shared datastores with matching zone tags.

List of required privileges

i To create a role and assign it to a user, refer to Configuration in vSphere Client and Configuration with govc sections.

A detailed list of privileges required for Deckhouse Platform to work in vSphere:

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

—	— (assigned by default when creating a role)	`System.Anonymous`
		`System.Read`
		`System.View`
			Basic access to vSphere Inventory objects required for all Deckhouse vSphere integration components.

Cns	Searchable	`Cns.Searchable`	Search and mapping of Container Native Storage objects when the CSI driver works with Kubernetes volumes.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Datastore	Allocate space,
	Browse datastore,
	Low level file operations
		`Datastore.AllocateSpace`
		`Datastore.Browse`
		`Datastore.FileManagement`
			Disk provisioning when creating virtual machines and ordering `PersistentVolumes` in the cluster.

Folder	Create folder,
	Delete folder,
	Move folder,
	Rename folder
		`Folder.Create`
		`Folder.Delete`
		`Folder.Move`
		`Folder.Rename`
			Grouping a Deckhouse Platform cluster in a single `Folder` in vSphere Inventory.

Global	Global tag, System tag	`Global.GlobalTag` `Global.SystemTag`	Access to global and system tags used by Deckhouse Platform when working with vSphere objects.

vSphere Tagging	Assign or Unassign vSphere Tag,
	Assign or Unassign vSphere Tag on Object,
	Create vSphere Tag,
	Create vSphere Tag Category,
	Delete vSphere Tag,
	Delete vSphere Tag Category,
	Edit vSphere Tag,
	Edit vSphere Tag Category,
	Modify UsedBy Field for Category,
	Modify UsedBy Field for Tag
		`InventoryService.Tagging.AttachTag`
		`InventoryService.Tagging.ObjectAttachable`
		`InventoryService.Tagging.CreateTag`
		`InventoryService.Tagging.CreateCategory`
		`InventoryService.Tagging.DeleteTag`
		`InventoryService.Tagging.DeleteCategory`
		`InventoryService.Tagging.EditTag`
		`InventoryService.Tagging.EditCategory`
		`InventoryService.Tagging.ModifyUsedByForCategory`
		`InventoryService.Tagging.ModifyUsedByForTag`
			Deckhouse Platform uses tags to identify the `Datacenter`, `Cluster`, and `Datastore` objects available to it, as well as to identify the virtual machines under its control.

Network	Assign network	`Network.Assign`	Connecting networks and port groups to Deckhouse Platform cluster virtual machines.

Resource	Assign virtual machine to resource pool,
	Create resource pool,
	Modify resource pool,
	Remove resource pool,
	Rename resource pool
		`Resource.AssignVMToPool`
		`Resource.CreatePool`
		`Resource.DeletePool`
		`Resource.EditPool`
		`Resource.RenamePool`
			Placement of Deckhouse Platform cluster virtual machines into the target resource pool and management of this pool.

VM Storage Policies (*Profile-driven Storage Privileges* in vSphere 7)	View VM storage policies (*Profile-driven storage view* in vSphere 7)	`StorageProfile.View`	Viewing storage policies used when creating virtual machines and dynamically provisioning volumes in the cluster.

vApp	Add virtual machine,
	Assign resource pool,
	Create,
	Delete,
	Import,
	Power Off,
	Power On,
	View OVF Environment,
	vApp application configuration,
	vApp instance configuration,
	vApp resource configuration
		`VApp.ApplicationConfig`
		`VApp.AssignResourcePool`
		`VApp.AssignVM`
		`VApp.Create`
		`VApp.Delete`
		`VApp.ExtractOvfEnvironment`
		`VApp.Import`
		`VApp.InstanceConfig`
		`VApp.PowerOff`
		`VApp.PowerOn`
		`VApp.ResourceConfig`
			Managing operations related to deployment and configuration of vApp and OVF templates used when creating virtual machines.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Change Configuration	Add existing disk,
	Add new disk,
	Add or remove device,
	Advanced configuration,
	Set annotation,
	Change CPU count,
	Toggle disk change tracking,
	Extend virtual disk,
	Acquire disk lease,
	Modify device settings,
	Configure managedBy,
	Change Memory,
	Query unowned files,
	Configure Raw device,
	Reload from path,
	Remove disk,
	Rename,
	Reset guest information,
	Change resource,
	Change Settings,
	Change Swapfile placement,
	Upgrade virtual machine compatibility
		`VirtualMachine.Config.AddExistingDisk`
		`VirtualMachine.Config.AddNewDisk`
		`VirtualMachine.Config.AddRemoveDevice`
		`VirtualMachine.Config.AdvancedConfig`
		`VirtualMachine.Config.Annotation`
		`VirtualMachine.Config.CPUCount`
		`VirtualMachine.Config.ChangeTracking`
		`VirtualMachine.Config.DiskExtend`
		`VirtualMachine.Config.DiskLease`
		`VirtualMachine.Config.EditDevice`
		`VirtualMachine.Config.ManagedBy`
		`VirtualMachine.Config.Memory`
		`VirtualMachine.Config.QueryUnownedFiles`
		`VirtualMachine.Config.RawDevice`
		`VirtualMachine.Config.ReloadFromPath`
		`VirtualMachine.Config.RemoveDisk`
		`VirtualMachine.Config.Rename`
		`VirtualMachine.Config.ResetGuestInfo`
		`VirtualMachine.Config.Resource`
		`VirtualMachine.Config.Settings`
		`VirtualMachine.Config.SwapPlacement`
		`VirtualMachine.Config.UpgradeVirtualHardware`
			Managing the lifecycle of Deckhouse Platform cluster virtual machines.

Virtual Machine > Edit Inventory	Create new,
	Create from existing,
	Remove,
	Move
		`VirtualMachine.Inventory.Create`
		`VirtualMachine.Inventory.CreateFromExisting`
		`VirtualMachine.Inventory.Delete`
		`VirtualMachine.Inventory.Move`
			Creating, deleting, and moving Deckhouse Platform cluster virtual machines in vSphere Inventory.

Virtual Machine > Guest Operations	Guest Operation Queries	`VirtualMachine.GuestOperations.Query`	Retrieving information from the guest operating system of virtual machines.

Virtual Machine > Interaction	Answer question,
	Device connection,
	Guest operating system management by VIX API,
	Power Off,
	Power On,
	Reset,
	Configure CD media,
	Install VMware Tools
		`VirtualMachine.Interact.AnswerQuestion`
		`VirtualMachine.Interact.DeviceConnection`
		`VirtualMachine.Interact.GuestControl`
		`VirtualMachine.Interact.PowerOff`
		`VirtualMachine.Interact.PowerOn`
		`VirtualMachine.Interact.Reset`
		`VirtualMachine.Interact.SetCDMedia`
		`VirtualMachine.Interact.ToolsInstall`
			Managing virtual machine power state, device connections, and interaction with the guest operating system.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Provisioning	Clone virtual machine,
	Customize guest,
	Deploy template,
	Allow virtual machine download,
	Allow virtual machine files upload,
	Read customization specifications
		`VirtualMachine.Provisioning.Clone`
		`VirtualMachine.Provisioning.Customize`
		`VirtualMachine.Provisioning.DeployTemplate`
		`VirtualMachine.Provisioning.GetVmFiles`
		`VirtualMachine.Provisioning.PutVmFiles`
		`VirtualMachine.Provisioning.ReadCustSpecs`
			Cloning virtual machine templates, customizing them, and deploying them when creating Deckhouse Platform cluster nodes.

Virtual Machine > Snapshot Management	Create snapshot,
	Remove Snapshot,
	Rename Snapshot
		`VirtualMachine.State.CreateSnapshot`
		`VirtualMachine.State.RemoveSnapshot`
		`VirtualMachine.State.RenameSnapshot`
			Managing snapshots of virtual machines and volumes in scenarios where this functionality is used by platform components.

Additional resources

- Module `cloud-provider-vsphere` provider configuration

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in VMware vSphere

Storage

The following storage types are used in VMware vSphere for Kubernetes cluster data:

- **Datastores**: Used to store the root disks of virtual machines;
- **CNS disks (Container Native Storage)**: Used for automatic creation of PersistentVolumes via CSI.

Deckhouse Platform (DP) automatically creates a StorageClass for each Datastore and DatastoreCluster that is tagged as a `zone`.

You can specify:

- The default StorageClass name (`default`).
- Exclusions via the `exclude` field in a form of a list of names or patterns for StorageClasses that should not be created.

Example configuration using ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    storageClass:
      default: fast-lun102
      exclude:
        - ".*-lun101-.*"
        - slow-lun103
```

Resizing a volume (PVCs)

DP supports Online Resize PersistentVolume starting with vSphere 7.0U2. However, due to CSI and vSphere API specifics, additional steps are required after resizing a PVC:

1. Run `d8 k cordon <node_name>`.
2. Delete the Pod that uses the PVC.
3. Wait for the resize operation to complete:
   - Ensure the PVC no longer has the `Resizing` condition.
   - It’s safe to ignore the `FileSystemResizePending` status.
4. Run `d8 k uncordon <node_name>`.

Load balancing

Options for organizing incoming traffic load balancing:

1. **Via an external load balancer**. If your infrastructure includes an external load balancer (for example, NSX-T), you can route traffic directly to the cluster's frontend nodes.
2. **Via MetalLB**. For fault-tolerant load balancing within the cluster, it is recommended that you use MetalLB in BGP mode. In this case:
   - Frontend nodes receive two network interfaces.
   - A dedicated VLAN is required for BGP traffic.
   - The network must provide DHCP and internet access.
   - IP addresses and BGP router ASNs must be specified.
   - A pool of IP addresses to be announced must be defined.

> **i** Make sure there is connectivity between BGP routers and frontend nodes in the dedicated VLAN.

CSI

The storage subsystem uses CNS disks by default, with support for online resizing.
Legacy mode with FCD disks is also supported. The subsystem behavior is configured via the `compatibilityFlag` parameter.

Important information on PVC size expansion

Due to specifics of the CSI volume-resizer and vSphere API, after increasing a PVC size, you must perform the following steps:

1. On the node hosting the Pod, run `d8 k cordon <node_name>`.
2. Delete the Pod.
3. Ensure the resize operation completed successfully — the PVC **must not** have the `Resizing` condition.

   > **i** The `FileSystemResizePending` condition is not an issue.

4. On the node hosting the Pod, run `d8 k uncordon <node_name>`

Datastore configuration

> **i** You can also tag **Datastore** objects through the **VMware vSphere Client** — follow Datastore configuration in the connection and authorization guide. The steps below use **`govc` only**.

> ⚠ For dynamic `PersistentVolume` provisioning, a `Datastore` must be available on **each** ESXi host (shared datastore).

For PersistentVolume to function correctly, the datastore must be accessible from all ESXi hosts.

Assign tags:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Additional resources

- Module `cloud-provider-vsphere` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with VMware vSphere services

Deckhouse Platform integrates with VMware vSphere infrastructure and uses VsphereInstanceClass resources to describe the specifications of virtual machines created as part of the Kubernetes cluster.

Key features:

- Provisioning and removal of virtual machines via the vCenter API.
- Node placement across multiple clusters (`zones`) and datacenters (`region`).
- Use of VM templates with `cloud-init`.
- Support for networks with DHCP, static addressing, and additional interfaces.
- Storage management: provisioning root disks and PVCs based on Datastore or CNS disks.
- Support for incoming traffic load balancing:
  - Via external load balancers.

- Via MetalLB (in BGP mode).

> **i** DP supports hybrid integration with VMware vSphere. For configuration details, see Hybrid cluster with vSphere section.

vSphere resource management

Removing CloudPermanent nodes in vSphere

Nodes of the `CloudPermanent` type are created based on the node group configuration specified in the `nodeGroups` section of the VsphereClusterConfiguration resource.

The `replicas` parameter defines the required number of virtual machines in the group. After the configuration is changed, run the `dhctl converge` command to start Terraform and bring the state of virtual machines in VMware vSphere in line with the specified number of replicas.

To reduce the number of nodes in a group, decrease the `replicas` value and run `dhctl converge`.

> **⚠** Do not remove the group definition from the `nodeGroups` section while the value of the `replicas` parameter is greater than zero.
>
> If the group definition is removed before the number of replicas is reduced to zero, the Terraform state may become inconsistent with the state of virtual machines and disks in VMware vSphere. As a result, a subsequent `dhctl converge` run may fail and require manual state recovery.

Reducing the number of nodes

To reduce the number of nodes in a `CloudPermanent` group:

1. Open the vSphere configuration for editing:

   ```
   d8 system edit provider-cluster-configuration
   ```

2. In the `nodeGroups` section, find the required group and decrease the value of the `replicas` parameter to the required number of nodes.

   For example, to reduce the number of nodes in the `worker` group from three to two, set `replicas: 2`:

   ```
   nodeGroups:
   - name: worker
     replicas: 2
     zones:
     - zone-a
     instanceClass:
       numCPUs: 4
       memory: 8192
       template: Templates/ubuntu-24.04
       datastore: datastore-1
       mainNetwork: network-1
   ```

   Save the changes.

3. In the DP installer container, apply the updated configuration:

   ```
   dhctl converge \
     --ssh-host <MASTER-NODE-IP-ADDRESS> \
     --ssh-user <USERNAME> \
     --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
   ```

   > **i** Use an installer container of the same edition and version as the cluster.

4. Wait for `dhctl converge` to complete and check the number of nodes in the group:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<GROUP-NAME>
   ```

Completely removing a node group

A `CloudPermanent` group must be removed in two stages. First, reduce the number of replicas to zero and wait for the nodes and virtual machines to be removed. You can then remove the group definition from the vSphere configuration.

To completely remove a `CloudPermanent` group:

1. Open the vSphere configuration for editing:

```
d8 system edit provider-cluster-configuration
```

2. In the `nodeGroups` section, find the group to be removed and set `replicas: 0`. Do not remove the group definition at this stage. For example:

   ```
   nodeGroups:
   - name: worker
     replicas: 0
     zones:
     - zone-a
     instanceClass:
       numCPUs: 4
       memory: 8192
       template: Templates/ubuntu-24.04
       datastore: datastore-1
       mainNetwork: network-1
   ```

   Save the changes.

3. In the DP installer container, apply the updated configuration:

   ```
   dhctl converge \
     --ssh-host <MASTER-NODE-IP-ADDRESS> \
     --ssh-user <USERNAME> \
     --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
   ```

4. Wait for `dhctl converge` to complete successfully and make sure that no nodes remain in the group:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<GROUP-NAME>
   ```

   The command must not return any nodes from the group being removed.

   Also use vSphere Client to make sure that the virtual machines associated with the group have been removed.

5. Open the cloud provider configuration again:

   ```
   d8 system edit provider-cluster-configuration
   ```

6. Remove the group definition from the `nodeGroups` section and save the changes.
7. Apply the configuration again:

   ```
   dhctl converge \
     --ssh-host <MASTER-NODE-IP-ADDRESS> \
     --ssh-user <USERNAME> \
     --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
   ```

8. Make sure that the NodeGroup object has been removed:

   ```
   d8 k get nodegroup <GROUP-NAME>
   ```

   The command must return a message stating that the requested resource was not found.

Additional resources

- Module `cloud-provider-vsphere` custom resources
- Module `cloud-provider-vsphere` provider configuration

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Connection and authorization in zVirt

For Deckhouse Platform to create virtual machines in zVirt, assign the required roles to the zVirt user, prepare a cloud image of the operating system and a virtual machine template, and pass the user login and password in the credentials Secret.

Requirements

Deckhouse Platform requires the following to work with zVirt:

- A working zVirt installation with an accessible API
- zVirt version `4.0`–`4.4`

- A user with the permissions listed in the “User permissions” section
- A storage domain with an uploaded cloud image of the operating system
- A virtual machine template prepared from the cloud image

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

User permissions

Deckhouse Platform does not require the `SuperUser` role for the zVirt user. It is enough to assign the following predefined zVirt roles to the user on the objects the cluster works with:

Role	Object	Purpose
`UserVmManager`	zVirt cluster from the `clusterID` parameter	Creating, starting, stopping, and deleting node virtual machines, creating their network interfaces, attaching and detaching disks
`UserTemplateBasedVm`	Virtual machine template	Creating virtual machines from the template
`DiskOperator`	Storage domains that contain the template disk and where disks for etcd and cluster volumes are created	Creating, resizing, attaching, detaching, and deleting disks
`VnicProfileUser`	vNIC profile	Creating a virtual machine network interface from the vNIC profile
`TagManager`	Entire system (system permission)	Creating the cluster tag and assigning it to virtual machines of CloudEphemeral nodes

The `TagManager` role is assigned only on the entire system, because zVirt checks the tag management permission at this level.

⚠ The `TagManager` role is administrative. With it, the user can view all objects via the zVirt API but can modify only the objects on which the roles from the table are assigned to the user.

zVirt assigns some roles to the Everyone group by default, so you do not need to assign them to the user separately:

- `UserTemplateBasedVm` on a template created with the “Allow all users to access this Template” checkbox selected
- `VnicProfileUser` on a public vNIC profile
- `CpuProfileOperator` on the cluster CPU profile
- `DiskProfileUser` on the disk profiles of storage domains

If these permissions were removed in your zVirt installation, assign the roles to the user.

To assign a role on an object:

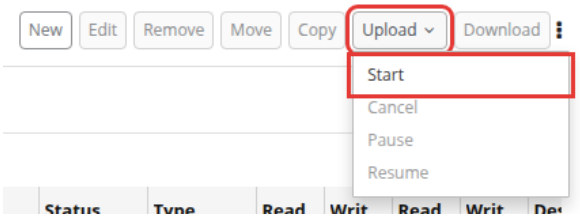
1. Open the object in the zVirt Administration Portal, for example, a cluster in the “Compute → Clusters” section.
2. Go to the “Permissions” tab and click “Add”.
3. Select the user and the role and save the permission.

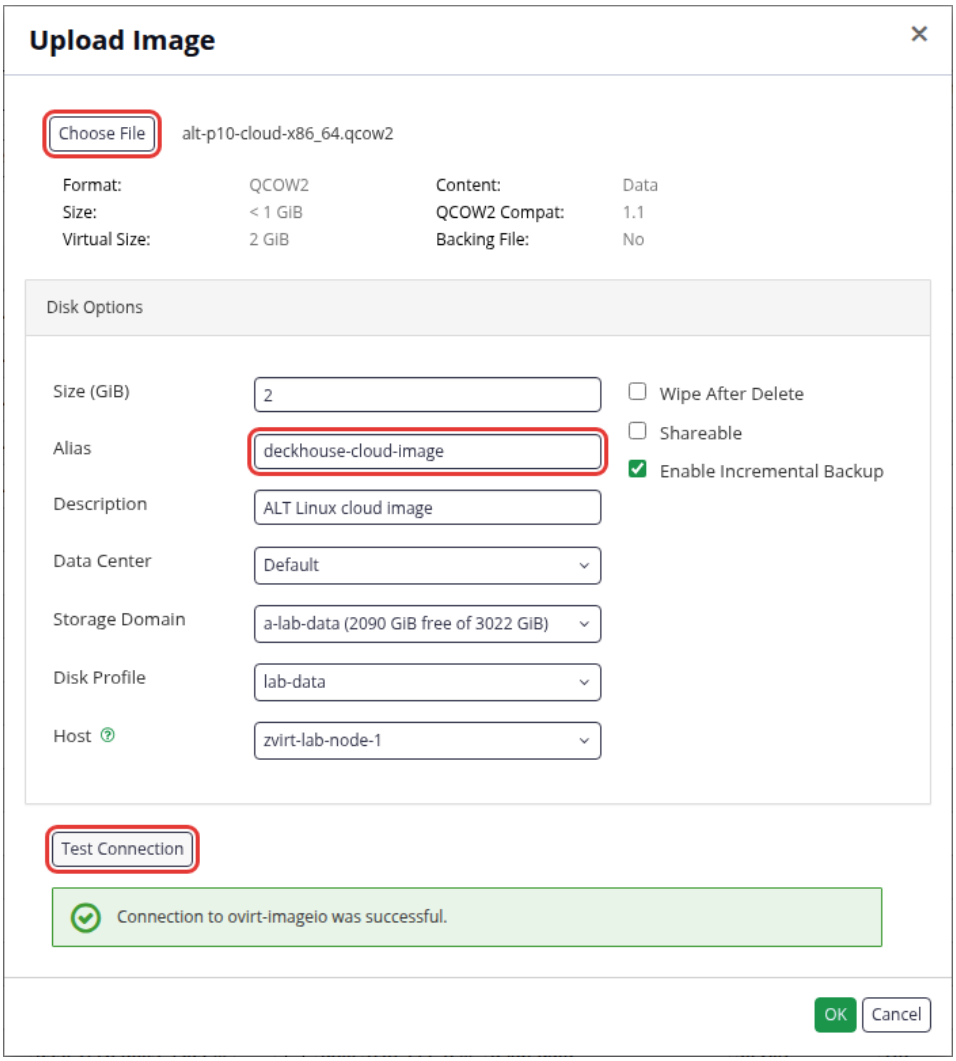
Assign the `TagManager` role in the “Administration → Configure” section on the “System Permissions” tab.

Preparing a cloud image

The cloud image of the operating system (OS) is uploaded to the zVirt disk storage. To upload the image:

1. Open the zVirt Administration Portal and go to the “Storage → Disks” section.
2. Upload the OS cloud image (for example, `.qcow2` or `.img`) to the storage.





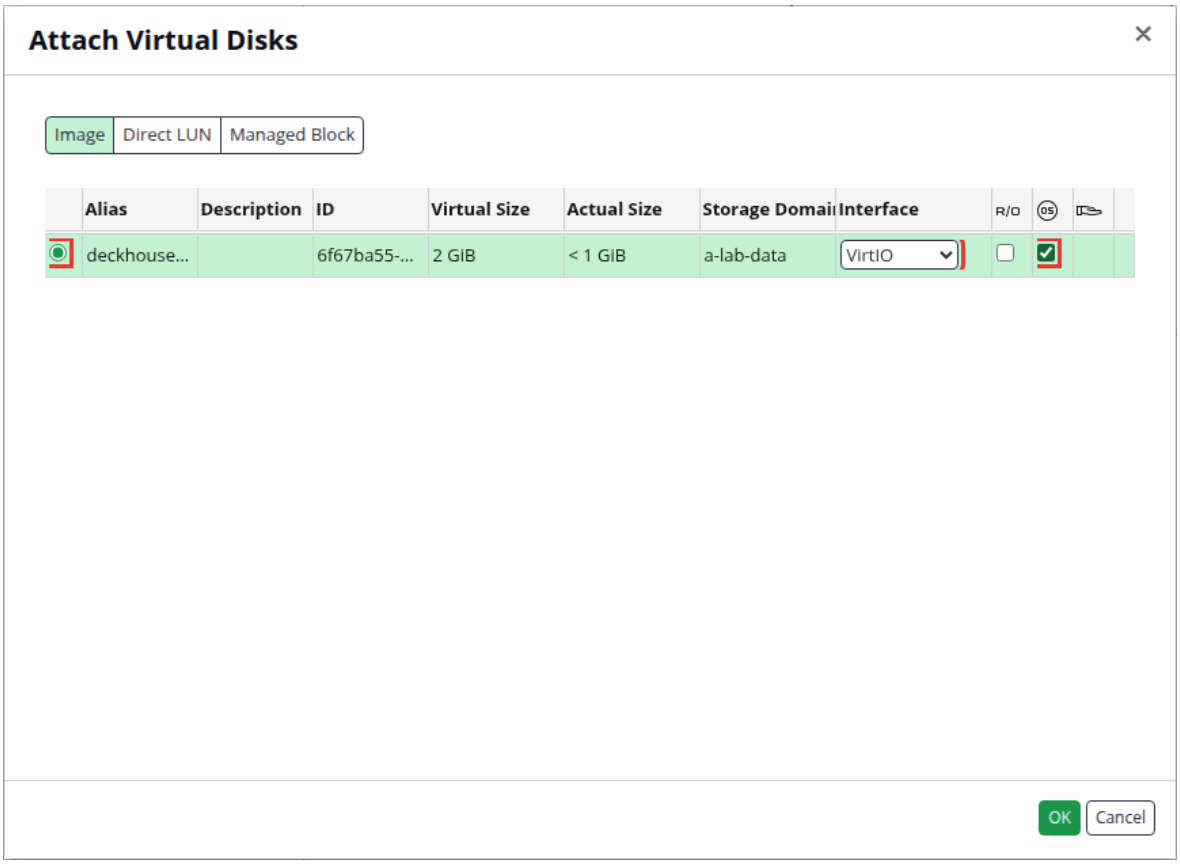
3. Wait for the upload to complete. The disk must get the `OK` status.



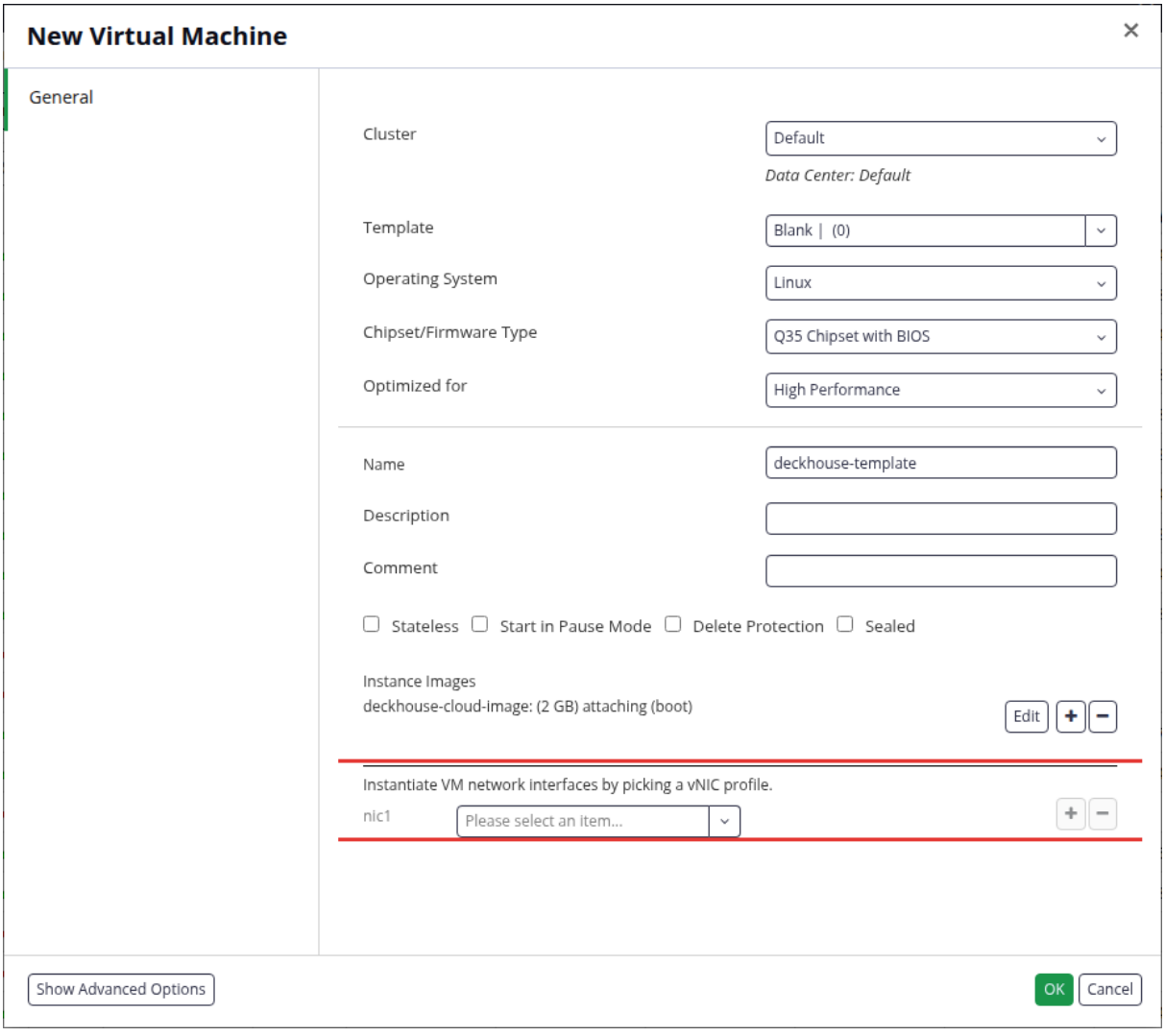
Preparing a virtual machine template

The template is created from a virtual machine (VM) with the OS cloud image attached. To prepare the template:

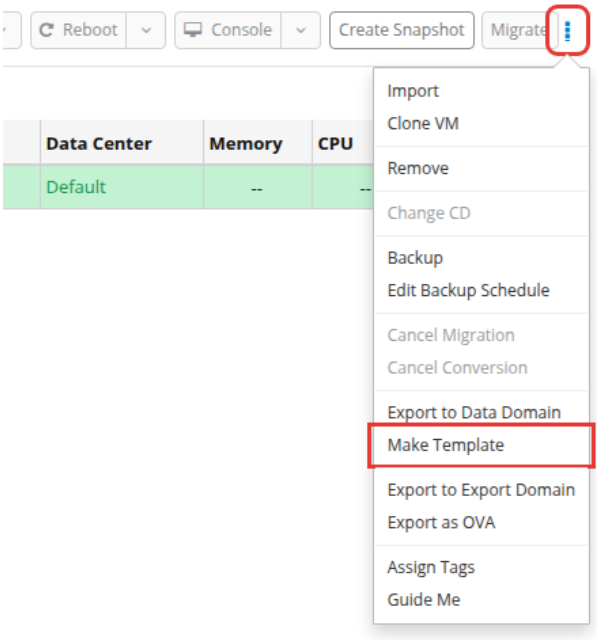
1. Open the “Compute → Virtual Machines” section and create a VM.
2. Set the following parameters:
   - “Template”: `Blank`.
   - “Operating System”: The OS of the uploaded cloud image.
   - “Optimized for”: “High Performance”.
   - “Instance Images”: Click “Attach”, select the uploaded cloud image, select the checkbox in the “OS” column to make the disk bootable, and select the “VirtIO” interface.



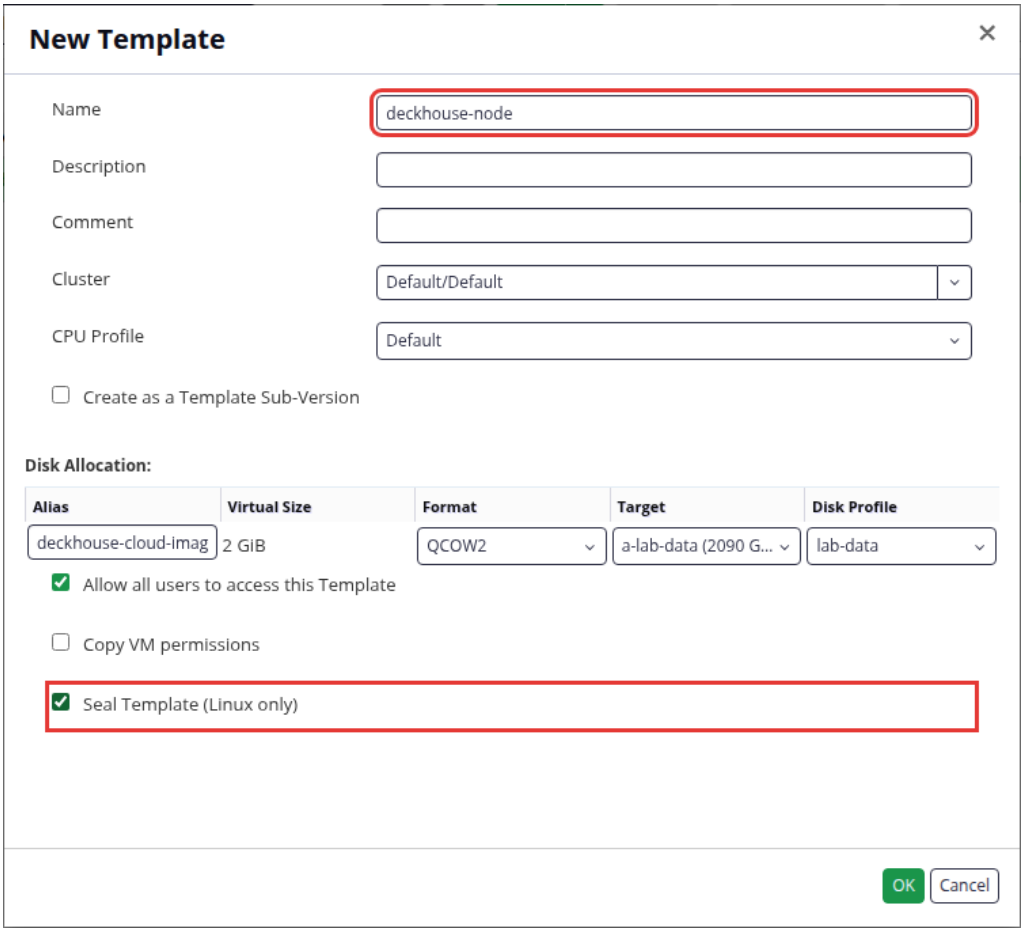
3. Do not add network interfaces. Deckhouse Platform creates a network interface when it creates a VM for a node.



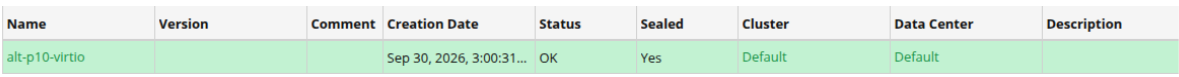
4. Leave the other parameters at their defaults and create the VM.
5. After the VM is created, select it in the list, open the additional actions menu (⋮), and select “Make Template”.



6. Specify the template name and enable the “Seal Template (Linux only)” parameter.



7. Make sure the template is displayed in the “Compute → Templates” section.



Replacing the OS image

The `template` parameter of ZvirtInstanceClass defines the image for `CloudEphemeral` nodes. With a ModuleConfig-based provider configuration, this parameter also defines the image for `CloudPermanent` nodes and master nodes.

With a ZvirtClusterConfiguration-based configuration, the image for `CloudPermanent` nodes and master nodes is set by `template` in `nodeGroups[].instanceClass` and `masterNodeGroup.instanceClass`, respectively.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?”.

Credentials Secret

The zVirt user login and password are stored in a separate Secret rather than in the ModuleConfig. The `cloud-provider-zvirt` module reads it and passes the credentials to its components that access the zVirt API.

The Secret is created during cluster installation together with the other resources of the initial configuration. The Secret must meet the following requirements:

- The name is `d8-credentials` and the namespace is `d8-cloud-provider-zvirt`.
- The type is `cloud-provider.deckhouse.io/credentials`.
- The `authScheme` field is set to `userPassword`.
- The `identity` field contains the zVirt user login.
- The `secret` field contains the zVirt user password.

```
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-zvirt
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: userPassword
  identity: <USERNAME>
  secret: <PASSWORD>
```

Replace `<USERNAME>` and `<PASSWORD>` with the zVirt user login and password.

A Secret that does not meet these requirements fails the check both during cluster installation and when applied in a running cluster.

To change the credentials, update the `identity` and `secret` fields:

```
d8 k -n d8-cloud-provider-zvirt edit secret d8-credentials
```

In clusters migrated to ModuleConfig, the login and password are moved to this Secret from the `provider.username` and `provider.password` parameters of the ZvirtClusterConfiguration resource.

Additional resources

- Global custom resources
- Module `cloud-provider-zvirt` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts and configuration in zVirt

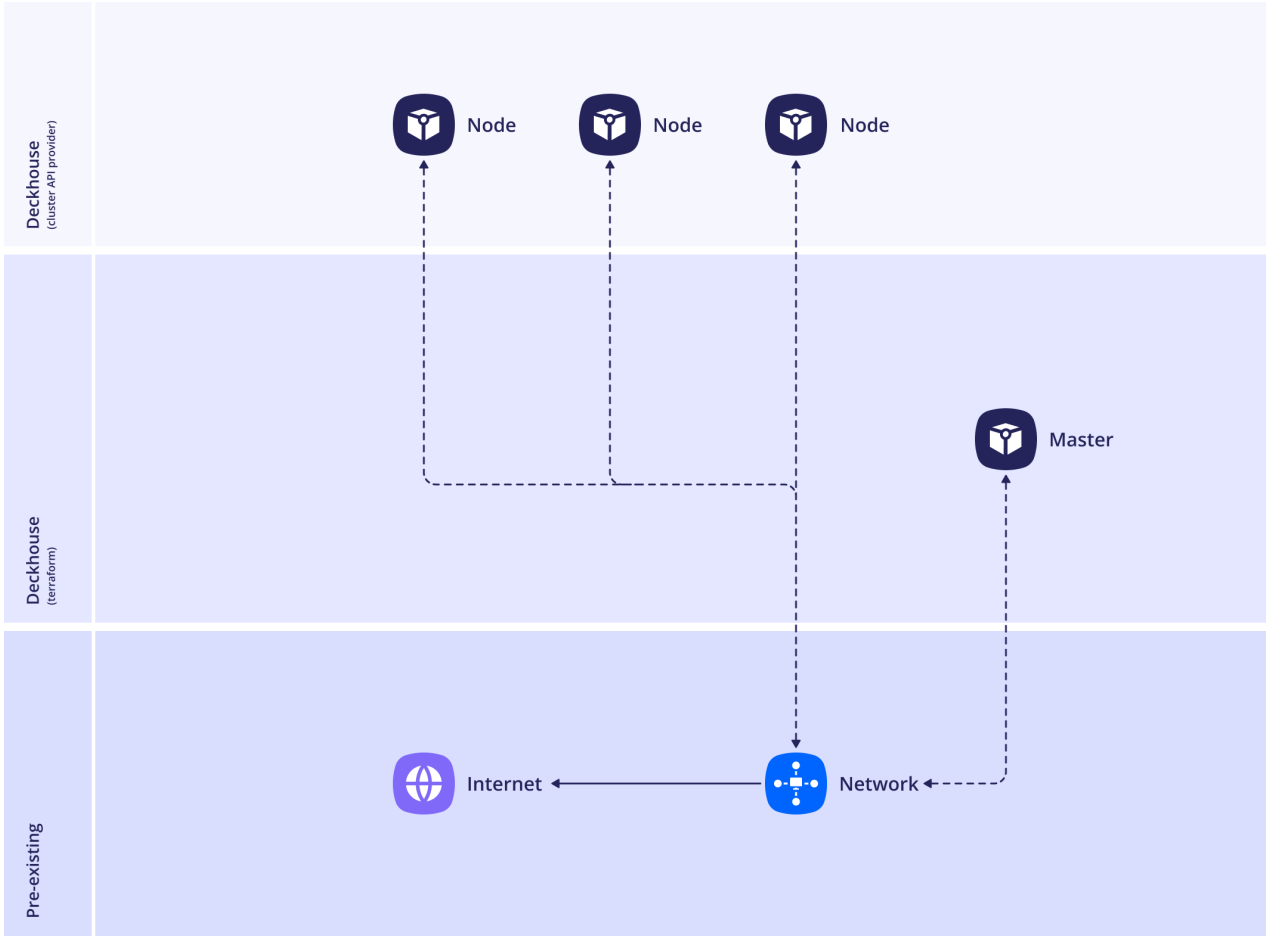
Layouts

The layout defines how cluster nodes are placed in the zVirt infrastructure. Only the Standard layout is supported. The layout is set with the `layout` parameter in the ModuleConfig resource of the `cloud-provider-zvirt` module.

Standard

In the Standard layout, all cluster nodes are created in a single zVirt cluster. Key features of the layout:

- Virtual machines are created in the zVirt cluster set with the `clusterID` parameter.
- The zVirt cluster must have shared storage domains available to all its hosts and CPUs of the same type.
- Virtual machines are cloned from a template prepared from a cloud image.
- The virtual machine network interface is created from the vNIC profile set with the `vnicProfileID` parameter of the ZvirtInstanceClass resource.
- Deckhouse Platform creates and deletes the node virtual machines via the zVirt API.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        server: <ZVIRT_API_URL>
        clusterID: <CLUSTER_ID>
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
```

Configuration

The zVirt cloud provider parameters are set in four resources:

- The ModuleConfig of the `cloud-provider-zvirt` module sets the layout, the zVirt API connection parameters, and the zVirt cluster to create virtual machines in.
- The `d8-credentials` Secret in the `d8-cloud-provider-zvirt` namespace stores the zVirt user login and password in the format covered in the “Credentials Secret” section.
- ZvirtInstanceClass describes the virtual machine parameters.
- NodeGroup describes the node groups and references the instance class. Master nodes are described by the `master` node group with the `CloudPermanent` type.

To change the provider settings in a running cluster, edit the ModuleConfig:

```
d8 k edit mc cloud-provider-zvirt
```

To apply the changes to the master nodes and the nodes of CloudPermanent groups, run the following command:

```
dhctl converge
```

⚠ If the cluster was installed with the ZvirtClusterConfiguration schema, it must be migrated to the ModuleConfig-based configuration. Until the migration is complete, the `D8CloudProviderZvirtMigrationPending` alert may fire and Deckhouse Platform updates may be blocked.

The migration procedure is described in “How to migrate a cloud provider to ModuleConfig-based configuration?”.

Configuration example

Below is a minimal configuration of a cluster with the Standard layout, one master node, and one group of worker nodes:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        server: <ZVIRT_API_URL>
        clusterID: <CLUSTER_ID>
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
---
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-zvirt
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: userPassword
  identity: <USERNAME>
  secret: <PASSWORD>
---
apiVersion: deckhouse.io/v1
kind: ZvirtInstanceClass
metadata:
  name: master
spec:
  numCPUs: 4
  memory: 8192
  rootDiskSizeGb: 50
  etcdDiskSizeGb: 10
  template: <TEMPLATE_NAME>
  vnicProfileID: <VNIC_PROFILE_ID>
  storageDomainID: <STORAGE_DOMAIN_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: ZvirtInstanceClass
      name: master
    minPerZone: 1
    maxPerZone: 1
---
apiVersion: deckhouse.io/v1
kind: ZvirtInstanceClass
metadata:
  name: worker
spec:
  numCPUs: 4
  memory: 8192
  template: <TEMPLATE_NAME>
```

```
  vnicProfileID: <VNIC_PROFILE_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: ZvirtInstanceClass
      name: worker
    minPerZone: 1
    maxPerZone: 3
```

Where:

- `ZVIRT API URL`: URL of the zVirt API endpoint (the `server` parameter), for example, `https://zvirt.example.com/ovirt-engine/api`.
- `CLUSTER ID`: ID of the zVirt cluster to create virtual machines in (the `clusterID` parameter).
- `SSH PUBLIC_KEY`: Public SSH key for accessing the nodes (the `sshPublicKey` parameter).
- `USERNAME` and `PASSWORD`: zVirt user login and password for the credentials Secret.
- `TEMPLATE_NAME`: Name of the virtual machine template prepared as described in the “Preparing a virtual machine template” section.
- `VNIC PROFILE ID`: ID of the vNIC profile used to create the virtual machine network interface.
- `STORAGE_DOMAIN_ID`: ID of the zVirt storage domain to create the etcd disk in.

The number of master nodes is set with the `minPerZone` parameter of the `master` node group. In the ZvirtInstanceClass resource of the `master` group, set the `etcdDiskSizeGb` and `storageDomainID` parameters, which set the size and the storage domain of a separate disk for etcd. The `etcdDiskSizeGb` parameter cannot be set in the instance classes of other CloudPermanent groups.

If the zVirt API uses a self-signed certificate, pass the CA certificate in the `caBundle` parameter or set the `insecure` parameter to `true`.

The other parameters are described in the module settings and in the ZvirtInstanceClass resource description.

> **i** The `clusterID`, `vnicProfileID`, and `storageDomainID` values can be obtained via the zVirt API or the Administration Portal. Getting the vNIC profile ID via the API is described in the “How to get the vNIC profile ID?” section.

Static network configuration of nodes

Nodes whose group has no static network configuration get their network settings over DHCP. To assign static IP addresses to the nodes of CloudPermanent groups, set the `customNetworkConfigs` parameter in the `nodes.parameters` section of the ModuleConfig resource. The parameter key sets the node group name, and the value sets the network interface configuration of the nodes in this group.

Example of the static network configuration for the `master` group of three nodes:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 2
  settings:
    nodes:
      parameters:
        customNetworkConfigs:
          master:
            networkInterfaceName: enp1s0
            networkInterfaceAddresses:
              - 192.168.1.10
              - 192.168.1.11
              - 192.168.1.12
            networkInterfaceNetmask: 255.255.255.0
            networkInterfaceGateway: 192.168.1.1
            dnsServers:
              - 192.168.1.2
```

Addresses from the `networkInterfaceAddresses` list are assigned to the group nodes in order, one address per node, so specify an address for each node of the group. Duplicate addresses in the list are not allowed.

⚠ If you change the network settings of existing nodes in the `customNetworkConfigs` parameter, `dhctl converge` recreates the virtual machines of these nodes.

Additional resources

- Global custom resources
- Module `cloud-provider-zvirt` documentation
- Module `node-manager` custom resources

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage and load balancing in zVirt

Storage

The `cloud-provider-zvirt` module provisions zVirt disks for cluster volumes via the `csi.ovirt.org` CSI driver. Volumes are requested with standard PersistentVolumeClaim resources using the StorageClasses the module creates.

StorageClasses for storage domains

For each active zVirt storage domain, the module creates a StorageClass whose name is derived from the storage domain name. Letters in the name are converted to lowercase, spaces are replaced with hyphens, and the other characters except Latin letters, digits, hyphens, and dots are removed. Hyphens and dots at the beginning and end of the name are also removed.

The created StorageClasses have the following properties:

- The disk for a volume is created in the storage domain the StorageClass corresponds to.
- Space for the disk is allocated in full at once (thin provisioning is not used).
- A volume is created when the workload that uses it is scheduled to a node (`volumeBindingMode: WaitForFirstConsumer`).
- The volume size can be increased (`allowVolumeExpansion: true`).

If the global `defaultClusterStorageClass` parameter is not set and the cluster has no other default StorageClass, the alphabetically first StorageClass of the module becomes the default StorageClass.

To prevent the module from creating StorageClasses for some storage domains, list the names of these StorageClasses in the `excludedStorageClasses` parameter of the ModuleConfig resource. Each item of the list is a regular expression matched against the whole StorageClass name.

Example of excluding StorageClasses:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 2
  settings:
    storage:
      parameters:
        excludedStorageClasses:
          - hostedstorage
          - backup-.*
```

In the example, the module does not create the `hostedstorage` StorageClass and the StorageClasses whose names start with `backup-`.

To disable disk provisioning in the cluster, set the `storage.disabled` parameter to `true`. In this case, the module does not deploy the CSI driver and does not create StorageClasses.

Virtual machine disks

The root disk of a virtual machine is created when the template is cloned, and then it is resized to the value of the `rootDiskSizeGb` parameter of the ZvirtInstanceClass resource.

On master nodes, a separate disk is created for etcd. Its size is set with the `etcdDiskSizeGb` parameter, and the storage domain is set with the `storageDomainID` parameter of the ZvirtInstanceClass resource of the `master` group. The storage domain must be available in the zVirt cluster set with the `clusterID` parameter.

Load balancing

The `cloud-provider-zvirt` module does not create load balancers for Service objects of the `LoadBalancer` type. To accept incoming traffic, use one of the following options:

- An external load balancer, for example, a hardware or software one, that forwards traffic to ports `80` and `443` of the cluster frontend nodes.
- The `metallb` module in L2 mode that announces load balancer IP addresses in the node network.

For L2 mode, allocate an IP address range in the network the frontend nodes are connected to via the vNIC profile from the `vnicProfileID` parameter. Addresses from this range must not be assigned by the network DHCP server. A configuration example is described in the “Example of metallb usage in L2 LoadBalancer mode” section.

> **i** Operation of the `metallb` module in BGP mode in zVirt is not guaranteed and depends on the network infrastructure.

Additional resources

- Global parameters
- Module `cloud-provider-zvirt` documentation
- Module `metallb` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with zVirt services

Deckhouse Platform integrates with the zVirt infrastructure and creates virtual machines for cluster nodes based on the definitions in the ZvirtInstanceClass resource.

Key features

The integration with zVirt provides the following features:

- Creating and deleting virtual machines in zVirt when node groups (NodeGroup) are created and scaled.
- Cloning virtual machines from a prepared template based on a cloud image with `cloud-init`.
- Creating the virtual machine network interface from a vNIC profile automatically.
- Getting node network settings over DHCP or the static network configuration for the nodes of CloudPermanent groups.
- Provisioning volumes via the CSI driver and StorageClasses for zVirt storage domains.
- Removing from the cluster the nodes whose virtual machines no longer exist in zVirt.

Additional resources

- Module `cloud-provider-zvirt` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid clusters

A hybrid cluster is a static Deckhouse Platform (DP) cluster extended with nodes placed in the infrastructure of a supported provider, such as Yandex Cloud, VMware Cloud Director or VMware vSphere.

In this scenario, the control plane and the initial cluster nodes remain part of the static DP cluster, while additional nodes are added through the corresponding provider module. These nodes can be created automatically through the provider API or connected manually if the virtual machines were created in advance.

This approach allows you to expand an existing static cluster without creating a separate Kubernetes cluster: increase compute capacity, place some workloads in another infrastructure, or gradually migrate services there. Applications still use a single Kubernetes control plane: the same API, shared resources, and unified mechanisms for scheduling, monitoring, updates, and operations.

Hybrid integration is performed on top of an already deployed static DP cluster (`clusterType: Static`). Then the corresponding cloud provider module is enabled, allowing DP to obtain information about the external infrastructure and create or connect additional nodes.

Depending on the provider, nodes can be added automatically through the provider API or connected manually by running a bootstrap script on pre-created virtual machines.

The general principles of hybrid integration are described on this page. The procedure for preparing the infrastructure and adding nodes depends on the provider being used and is described in separate guides:

- Yandex Cloud;
- VMware Cloud Director;
- VMware vSphere.

NodeGroup types

In a hybrid scenario, DP uses the following NodeGroup types:

- `CloudEphemeral` — nodes that DP creates and deletes automatically through the configured cloud provider API.
- `CloudStatic` — nodes that are created manually by the user or by external tools in the same cloud infrastructure for which cloud provider integration is configured. CSI runs on such nodes, and the Node object is managed by cloud-controller-manager: it is automatically enriched with zone and region information based on provider data.

Ways to add nodes

Nodes from the provider infrastructure can be connected in the following ways:

- **Automatic creation of cloud nodes**. DP creates virtual machines through the provider API. VM parameters are described by the `*InstanceClass` resource, while the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created cloud nodes using a bootstrap script**. A virtual machine is created in advance by the user in the provider infrastructure, after which a DP bootstrap script is run on it to connect the node to the cluster. This scenario uses a NodeGroup with the `CloudStatic` type. Such a node is managed by the corresponding provider's cloud-controller-manager.

This section describes the general network requirements for hybrid clusters. Provider-specific requirements, including network parameters, virtual machine templates, credentials, placement parameters, and ways to add nodes, are described in separate guides for Yandex Cloud, VCD, and vSphere.

General network requirements

Two-way network connectivity sufficient for DP and Kubernetes components must be configured between the nodes of the initial static cluster and the connected nodes placed in the provider infrastructure.

Connectivity must provide:

- access from connected nodes to the Kubernetes API of the initial cluster;
- access from connected nodes to DNS servers;
- access from connected nodes to the container image registry and other external services required to download images and packages;
- access from the control plane and system components to the connected nodes for kubelet, networking components, monitoring, and Kubernetes service operations;
- access from DP components that interact with the provider infrastructure to the corresponding provider API.

The full list of required connections is provided in the Network interaction section, and access restriction recommendations are provided in the Configuring network policies section.

Additionally, it is recommended to check:

- routing between the networks of the initial static cluster and the connected nodes in both directions;
- consistent MTU values across the entire network path, especially when tunnels are used;
- traffic encapsulation parameters when using Cilium, including `tunnelMode`, if traffic filtering is applied between sites.

Specific requirements for networks, subnets, virtual machine templates, credentials, and additional parameters depend on the infrastructure provider being used and are described in the "Prerequisites" section for the corresponding provider.

Additional resources

- Global custom resources
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid cluster with Yandex Cloud

The following describes the process of adding nodes from Yandex Cloud to an existing static Deckhouse Platform cluster.

Integration with Yandex Cloud uses the `cloud-provider-yandex` module. It provides interaction between DP and the Yandex Cloud API, retrieval of information about cloud infrastructure, creation of virtual machines, work with network parameters, and connection of nodes to an existing cluster.

This section describes two ways to add nodes:

- **Automatic node creation in Yandex Cloud**. DP creates virtual machines through the Yandex Cloud API. VM parameters are defined by the YandexInstanceClass resource, and the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DP bootstrap script. This scenario uses NodeGroup with the `CloudStatic` type.

Prerequisites

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter, and its nodes meet the minimum system requirements.
- "Network connectivity" is configured between the network of static nodes and the Yandex Cloud VPC. Yandex Cloud nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the

"Network interaction" and "Network policy configuration" sections. When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.
- The requirements from the "Connection and authorization in Yandex Cloud" section are met:
  - A service account is prepared.
  - A folder where resources will be created is selected.
  - The required roles and access to the VPC being used are configured.

Adding automatically created nodes

To run the preparation commands, you need the Yandex Cloud CLI (`yc`). You can use it on the administrator's workstation. The `yc` CLI is not required on the cluster master node: only the prepared manifests need to be applied in the cluster.

1. Get the identifiers of the cloud and folder where nodes will be created:

   ```
   yc resource-manager cloud list
   yc resource-manager folder list
   ```

2. Specify the obtained identifiers in variables:

   ```
   export CLOUD_ID="<CLOUD_ID>"
   export FOLDER_ID="<FOLDER_ID>"
   ```

   Where:

   - `CLOUD ID` — Yandex Cloud cloud ID;
   - `FOLDER_ID` — ID of the folder where resources will be created.

3. Get the identifiers of the network, subnet, and zone where nodes will be created:

   ```
   yc vpc network list --folder-id "$FOLDER_ID"
   yc vpc subnet list --folder-id "$FOLDER_ID"
   yc compute zone list
   ```

4. Specify the obtained values in variables:

   ```
   export NETWORK_ID="<NETWORK_ID>"
   export SUBNET_ID="<SUBNET_ID>"
   export ZONE="<ZONE>"
   export ROUTE_TABLE_ID="<ROUTE_TABLE_ID>"
   ```

   Where:

   - `NETWORK ID` — VPC network ID;
   - `SUBNET_ID` — ID of the subnet where nodes will be created;
   - `ZONE` — availability zone that corresponds to the selected subnet, for example `ru-central1-a`;
   - `ROUTE TABLE ID` — ID of the route table attached to the subnet. It is shown in the `ROUTE TABLE ID` column of the `yc vpc subnet list` output.

   For details, see "Connecting and authorizing in Yandex Cloud".

5. Create a service account in the required Yandex Cloud folder and assign permissions to it:

   ```
   yc iam service-account create \
     --name dkp-hybrid \
     --folder-id "$FOLDER_ID"

   export SA_ID="$(yc iam service-account get \
     --name dkp-hybrid \
     --folder-id "$FOLDER_ID" \
     --format json | jq -r .id)"

   yc resource-manager folder add-access-binding "$FOLDER_ID" \
     --role editor \
     --subject "serviceAccount:${SA_ID}"

   yc resource-manager folder add-access-binding "$FOLDER_ID" \
     --role vpc.admin \
     --subject "serviceAccount:${SA_ID}"
   ```

   The `editor` role is required to create and manage cloud resources, and the `vpc.admin` role grants access to VPC network resources.

6. Create a service account key and save it to a JSON file:

   ```
   yc iam key create \
     --service-account-id "$SA_ID" \
     --output dkp-hybrid-sa-key.json
   ```

Save the service account JSON key to the `SERVICE_ACCOUNT_JSON` environment variable in single-line format:

```
export SERVICE_ACCOUNT_JSON="$(jq -c . dkp-hybrid-sa-key.json)"
```

7. Save the administrator's public SSH key to the `SSH_PUBLIC_KEY` environment variable:

   ```
   export SSH_PUBLIC_KEY="$(cat ~/.ssh/id_rsa.pub)"
   ```

   If you use another key, specify its path instead of `~/.ssh/id_rsa.pub`.

   ⚠ The `SSH_PUBLIC_KEY` variable must contain the administrator's public SSH key that will be used to access the nodes being created. Do not use the public key from the service account JSON file.

8. Get the ID of the operating system image that will be used to create virtual machines and save it to the `IMAGE_ID` environment variable:

   ```
   export IMAGE_ID="$(yc compute image get-latest-from-family ubuntu-2404-lts \
     --folder-id standard-images \
     --format json | jq -r .id)"
   ```

   ⚠ The `IMAGE_ID` variable must contain the OS image ID in Yandex Cloud. Do not use an existing virtual machine ID or a service account key ID.

9. Specify the CIDR of the network where Yandex Cloud nodes will be placed:

   ```
   export NODE_NETWORK_CIDR="<NODE_NETWORK_CIDR>"
   ```

   `NODE_NETWORK_CIDR` is the CIDR that includes the internal IP addresses of Yandex Cloud nodes. For a single zone, it usually matches the CIDR of the selected subnet. For example, if nodes are created in the `10.128.0.0/24` subnet, specify `10.128.0.0/24`. You can get the subnet CIDR with the following command:

   ```
   yc vpc subnet list --folder-id "$FOLDER_ID"
   ```

10. Create the ModuleConfig manifest for the `cloud-provider-yandex` module. For example, `yandex-module-config.yaml`:

    ```
    cat > yandex-module-config.yaml <<EOF
    apiVersion: deckhouse.io/v1alpha1
    kind: ModuleConfig
    metadata:
      name: cloud-provider-yandex
    spec:
      version: 2
      enabled: true
      settings:
        provider:
          parameters:
            cloudID: ${CLOUD_ID}
            folderID: ${FOLDER_ID}
        nodes:
          parameters:
            layout: WithoutNAT
            nodeNetworkCIDR: ${NODE_NETWORK_CIDR}
            sshPublicKey: '${SSH_PUBLIC_KEY}'
            existingNetworkID: ${NETWORK_ID}
            existingRouteTableID: ${ROUTE_TABLE_ID}
            existingZoneToSubnetIDMap:
              ${ZONE}: ${SUBNET_ID}
            zones:
              - ${ZONE}
    EOF
    ```

    The manifest automatically uses the values of the environment variables set in the previous steps: `CLOUD_ID`, `FOLDER_ID`, `NODE_NETWORK_CIDR`, `SSH_PUBLIC_KEY`, `NETWORK_ID`, `ZONE`, and `SUBNET_ID`.

    The `existingNetworkID`, `existingZoneToSubnetIDMap`, and `existingRouteTableID` parameters describe the Yandex Cloud resources that already exist, because in a hybrid cluster DP does not create the network, the subnets, or the route table. The route table is required because the cloud controller manager creates the pod subnet routes in it and does not start with an empty value.

    The layout is set by the `layout` parameter. The available layouts are described in "Configuration and layouts".

11. Copy the `yandex-module-config.yaml` file to the cluster master node. Apply the manifest:

    ```
    d8 k apply -f yandex-module-config.yaml
    ```

    While DP is enabling the module, the validating webhooks of DP and the provider may be unavailable, and the command fails with `failed calling webhook`. This state is temporary, repeat the command in half a minute. The same applies to the next step with the Secret.

12. Wait until the module creates the `d8-cloud-provider-yandex` namespace:

    ```
    d8 k get namespace d8-cloud-provider-yandex
    ```

13. Create the Secret with the service account key:

    ```
    d8 k create -f - <<EOF
    apiVersion: v1
    kind: Secret
    metadata:
      name: d8-credentials
      namespace: d8-cloud-provider-yandex
    type: cloud-provider.deckhouse.io/credentials
    stringData:
      authScheme: serviceAccount
      secret: '${SERVICE_ACCOUNT_JSON}'
    EOF
    ```

    The Secret format is described in “Credentials Secret”.

14. Wait until the `cloud-provider-yandex` module is enabled and the YandexInstanceClass resource appears:

    ```
    d8 k get module cloud-provider-yandex -o wide
    d8 k get crd yandexinstanceclasses.deckhouse.io
    ```

15. Create a file with YandexInstanceClass and NodeGroup manifests. For example, `yandex-instanceclass-nodegroup.yaml`:

    ```
    apiVersion: deckhouse.io/v1
    kind: YandexInstanceClass
    metadata:
      name: yc-worker
    spec:
      cores: 4
      memory: 8192
      diskSizeGB: 50
      diskType: network-ssd
      mainSubnet: <SUBNET_ID>
    ---
    apiVersion: deckhouse.io/v1
    kind: NodeGroup
    metadata:
      name: yc-worker
    spec:
      nodeType: CloudEphemeral
      cloudInstances:
        classReference:
          kind: YandexInstanceClass
          name: yc-worker
        minPerZone: 1
        maxPerZone: 1
        zones:
          - ru-central1-a
    ```

    Where:

    - YandexInstanceClass describes the parameters of the virtual machine that will be created in Yandex Cloud;
    - `mainSubnet` — ID of the subnet from which the created nodes must have access to the static cluster nodes;
    - NodeGroup describes the node group that DP must maintain in the cluster;
    - `nodeType: CloudEphemeral` means that nodes will be created automatically through the cloud provider;
    - `cloudInstances.zones` must contain zones from the `zones` list in the module settings.

    In the `WithoutNAT` layout, the created nodes get a public IP address. To keep the nodes of the group on internal addresses only, set the `assignPublicIPAddress: false` parameter in the YandexInstanceClass and provide the nodes with another way out to the internet, for example through a subnet NAT gateway.

16. Apply the manifest:

```
d8 k apply -f yandex-instanceclass-nodegroup.yaml
```

After the manifest is applied, DP will start creating a virtual machine in Yandex Cloud through machine-controller-manager.

17. Check that the node appears in the cluster:

    ```
    d8 k get nodes -o wide
    ```

    Example expected output:

    ```
    NAME                                STATUS   ROLES                  AGE   VERSION    INTERNAL-
    IP
    static-master-0                     Ready    control-plane,master   1h    v1.33.10
    10.128.0.15
    yc-worker-f3564dca-7fc59-s2w5d      Ready    yc-worker              10m   v1.33.10
    10.128.0.21
    ```

18. To diagnose the state and troubleshoot possible issues, check the machine-controller-manager logs:

    ```
    d8 k -n d8-cloud-instance-manager get machinedeployments.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager get machinesets.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager get machines.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
    ```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-yandex` module is enabled:

  ```
  d8 k get module cloud-provider-yandex -o wide
  ```

- The `cloud-provider-yandex` module components are in the `Running` state:

  ```
  d8 k -n d8-cloud-provider-yandex get pods -o wide
  ```

- A virtual machine that will be connected to the cluster has been created in Yandex Cloud.
- The virtual machine is connected to the Yandex Cloud network and subnet used for hybrid integration with the cluster.
- The virtual machine has a network interface in the Yandex Cloud VPC network and subnet used for hybrid integration with the cluster. The IP address of this interface must belong to the CIDR specified in `nodeNetworkCIDR` and be reachable from the static cluster nodes.
- The virtual machine name in Yandex Cloud matches the hostname inside the operating system.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Check the virtual machine metadata in Yandex Cloud.

   The VM metadata must have `cloud-init` configured with the user that will be used for SSH connection.

   Example metadata:

   ```
   #cloud-config
   datasource:
     Ec2:
       strict_id: false
   ssh_pwauth: no
   users:
     - name: <USER>
       sudo: ALL=(ALL) NOPASSWD:ALL
       shell: /bin/bash
       ssh_authorized_keys:
         - <SSH_PUBLIC_KEY>
   ```

   Where:

   - `<USER>`: Username for SSH access to the virtual machine.
   - `<SSH_PUBLIC_KEY>`: Administrator’s public SSH key.

2. On the master node, create a file with the NodeGroup manifest and specify the node group name. In this example and the following steps, the `yc-manual` name is used. For example, `yandex-manual-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: yc-manual
spec:
  nodeType: CloudStatic
EOF
```

3. Make sure that the NodeGroup has been created and synchronized:

   ```
   d8 k get nodegroup yc-manual
   d8 k describe nodegroup yc-manual
   ```

   Example expected output:

   ```
   NAME        TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX   STANDBY
   STATUS   AGE   SYNCED
   yc-manual   CloudStatic   0       0       0
   1m    True
   ```

4. On the master node, get the bootstrap script for the created NodeGroup:

   ```
   NODE_GROUP=yc-manual

   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
     -o jsonpath='{.data.bootstrap\.sh}' > ${NODE_GROUP}-bootstrap.b64
   ```

5. Copy the bootstrap script to the VM being connected. If SSH access to the VM is available from the master node, run the following on the master node:

   ```
   scp ${NODE_GROUP}-bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
   ```

   > **i** To copy and run the bootstrap script, use the user specified in the VM metadata.

   If SSH access to the VM is available only from the administrator’s workstation, first copy the file from the master node to the workstation, and then from the workstation to the VM:

   ```
   scp <MASTER_USER>@<MASTER_IP>:/root/${NODE_GROUP}-bootstrap.b64 ./bootstrap.b64
   scp ./bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
   ```

   Where:

   - `<MASTER_USER>`: User for SSH access to the master node.
   - `<MASTER_IP>`: IP address of the master node.
   - `<USER>`: User on the VM being connected.
   - `<NODE_PUBLIC_OR_INTERNAL_IP>`: Public or internal IP address of the VM being connected.

6. On the VM being connected, decode the bootstrap script, set permissions, and run it:

   ```
   base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
   chmod +x /tmp/bootstrap.sh

   sudo bash /tmp/bootstrap.sh
   ```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

   ```
   d8 k get nodes -o wide
   ```

   Example expected output:

   ```
   NAME                    STATUS   ROLES       AGE   VERSION    INTERNAL-IP   EXTERNAL-IP
   static-master-0         Ready    master      1h    v1.33.10   10.128.0.15   <none>
   yandex-worker-hybrid    Ready    yc-manual   5m    v1.33.10   10.128.0.17   <PUBLIC_IP>
   ```

8. If connection fails, check the NodeGroup status, events, and component logs:

```
d8 k get nodegroup yc-manual
d8 k describe nodegroup yc-manual
d8 k describe node yandex-worker-hybrid
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Additional resources

- Global custom resources
- Module `cloud-provider-yandex` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid cluster with VCD

The following describes the process of adding nodes from VMware Cloud Director (VCD) to an existing static Deckhouse Platform (DP) cluster.

Integration with VCD uses the `cloud-provider-vcd` module. It provides interaction between DP and VMware Cloud Director, creation and deletion of virtual machines, retrieval of information about the VCD infrastructure, and integration with StorageClass and other provider capabilities.

This section describes two ways to add nodes:

- **Automatic node creation in VCD**. DP creates virtual machines through the VCD API. VM parameters are defined by the VCDInstanceClass resource, and the required number of nodes is defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DP bootstrap script. This scenario uses the NodeGroup resource with the `CloudStatic` type.

Prerequisites for VCD

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of static nodes and the network of virtual machines in VCD. VCD nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the Network interaction and Network policy configuration sections. When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.
- The requirements from the Connection and authorization in VMware vCloud Director section are met:
  - A tenant with allocated resources is configured in VCD.
  - A VCD account with a static password and administrator permissions is prepared.
  - A working network with an enabled DHCP server is configured in VCD.
  - The required VCD resources are prepared: VDC, vApp, templates, policies, and other parameters.

Adding automatically created nodes

1. Create a file with a ModuleConfig resource. For example, `cloud-provider-vcd-mc.yaml`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cloud-provider-vcd
   spec:
     version: 1
     enabled: true
     settings:
       mainNetwork: <NETWORK_NAME>
       organization: <ORGANIZATION>
       virtualDataCenter: <VDC_NAME>
       virtualApplicationName: <VAPP_NAME>
       sshPublicKey: <SSH_PUBLIC_KEY>
       provider:
         server: <API_URL>
         username: <USER_NAME>
         password: <PASSWORD>
         insecure: false
   ```

   Where:

   - `mainNetwork`: Name of the network where cloud nodes will be placed in VCD.
   - `organization`: Organization name in VCD.
   - `virtualDataCenter`: Virtual Data Center name in VCD.

- `virtualApplicationName`: Name of the vApp where nodes will be created, for example `dkp-vcd-app`.
- `sshPublicKey`: Public SSH key for accessing the nodes.
- `provider.server`: VCD API URL.
- `provider.username`: VCD username.
- `provider.password`: VCD user password.
- `provider.insecure`: Set to `true` if VCD uses a self-signed TLS certificate.

If a token is used for authentication, specify `apiToken` instead of `username` and `password`:

```
provider:
  server: <API_URL>
  apiToken: <API_TOKEN>
  insecure: false
```

2. Apply the ModuleConfig:

```
d8 k apply -f cloud-provider-vcd-mc.yaml
d8 k get mc cloud-provider-vcd
```

3. Make sure that all pods in the `d8-cloud-provider-vcd` namespace are in the `Running` state:

```
d8 k get pods -n d8-cloud-provider-vcd
```

4. Make sure that StorageClasses for VCD have been created in the cluster:

```
d8 k get sc
```

5. Create a file with the VCDInstanceClass and NodeGroup resources. For example, `vcd-instanceclass-nodegroup.yaml`:

```
---
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: worker
spec:
  rootDiskSizeGb: 50
  sizingPolicy: <SIZING_POLICY>
  storageProfile: <STORAGE_PROFILE>
  template: <VAPP_TEMPLATE>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VCDInstanceClass
      name: worker
    maxPerZone: 2
    minPerZone: 1
  nodeTemplate:
    labels:
      node-role/worker: ""
```

6. Apply the manifest:

```
d8 k apply -f vcd-instanceclass-nodegroup.yaml
```

After the manifest is applied, DP will start creating virtual machines in VCD managed by the `node-manager` module.

7. Make sure that the required number of nodes has appeared in the cluster:

```
d8 k get nodes -o wide
```

8. If VM creation fails, check the Machine and MachineSet objects and the machine-controller-manager logs:

```
d8 k -n d8-cloud-instance-manager get machinesets,machines -o wide
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-vcd` module is enabled:

```
d8 k get module cloud-provider-vcd -o wide
```

- The `cloud-provider-vcd` module components are in the `Running` state:

```
d8 k -n d8-cloud-provider-vcd get pods -o wide
```

- StorageClasses for VCD have been created in the cluster:

```
d8 k get sc
```

- A virtual machine that will be connected to the cluster has been created in VCD.
- The virtual machine name in VCD matches the hostname inside the operating system.
- The following value is set in the VM advanced parameters in VCD:

```
disk.EnableUUID = TRUE
```

- The virtual machine is connected to the VCD network used as the main network for the cluster's cloud nodes. Usually, this is the network specified in the `mainNetwork` parameter of the `cloud-provider-vcd` configuration or in the VCDInstanceClass being used.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a file with a NodeGroup resource and the CloudStatic node type. For example, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

2. Make sure that the NodeGroup has been created and synchronized:

```
d8 k get nodegroup cloud-static
```

   Example expected output:

```
NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
STANDBY   STATUS   AGE   SYNCED
cloud-static   CloudStatic   0       0       0
1m    True
```

3. Get the bootstrap script for the created NodeGroup:

```
NODE_GROUP=cloud-static

d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
  -o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

4. Copy the bootstrap script to the virtual machine being connected:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

5. Connect to the virtual machine over SSH:

```
ssh <USER>@<NODE_IP>
```

6. On the virtual machine, set permissions and run the bootstrap script:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

```
d8 k get nodes -o wide
```

Example expected output:

```
NAME                      STATUS   ROLES          AGE   VERSION    INTERNAL-IP
static-master-0           Ready    master         1h    v1.33.10   192.168.240.138
cloud-static-worker-0     Ready    cloud-static   5m    v1.33.10   192.168.240.151
```

Additional resources

- Global custom resources
- Module `cloud-provider-vcd` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid cluster with vSphere

The following describes the process of adding nodes from vSphere to an existing static Deckhouse Platform (DP) cluster.

Integration with vSphere uses the `cloud-provider-vsphere` module. It provides interaction between DP and vCenter, retrieval of information about virtual machines, work with placement parameters, and integration with vSphere infrastructure capabilities.

This section describes two ways to add nodes:

- **Automatic node creation in vSphere**. DP creates virtual machines through the vSphere API. VM parameters are defined by the VsphereInstanceClass resource, and the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DP bootstrap script. This scenario uses the NodeGroup resource with the `CloudStatic` type.

Prerequisites for vSphere

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of static nodes and the network of virtual machines in vSphere.
- vSphere nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the Network interaction and Network policy configuration sections.
- The requirements from the Connection and authorization in VMware vSphere section are met:
  - Access to vCenter is configured.
  - The vSphere account with the required privileges is prepared.
  - A virtual machine template is prepared.
  - Networks, Datastore, region tags, and zone tags are configured.
- When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.

Adding automatically created nodes

To connect an already running static cluster to vCenter, use the ModuleConfig resource of the `cloud-provider-vsphere` module.

In the `spec.settings` parameter, specify access parameters for vCenter, network settings, region and zone tags, and SSH keys that will be added to the created virtual machines.

An example configuration and description of the available parameters are provided in the module examples and in the section describing the module settings.

1. Create a file with ModuleConfig for the `cloud-provider-vsphere` module. For example, `vsphere-mc.yaml`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: "<VCENTER_FQDN>"
    username: "<USERNAME@DOMAIN.LOCAL>"
    password: "<PASSWORD>"
    insecure: true
    vmFolderPath: "<FOLDER_PATH_UNDER_DATACENTER>"
    regionTagCategory: "<TAG_CATEGORY_FOR_REGION>"
    zoneTagCategory: "<TAG_CATEGORY_FOR_ZONE>"
    region: "<REGION_TAG_NAME_ON_DATACENTER>"
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
    internalNetworkNames:
      - "<PORT_GROUP_NAME_FOR_INTERNAL_IP>"
    sshKeys:
      - "<SSH_PUBLIC_KEY_ONE_LINE>"
```

Parameter values:

- `host`: vCenter address.
- `username`, `password`: vSphere user credentials.
- `insecure`: Disables verification of the vCenter TLS certificate.
- `vmFolderPath`: Folder where virtual machines will be created.
- `regionTagCategory`, `zoneTagCategory`: Region and zone tag categories.
- `region`: Region tag.
- `zones`: List of zones where nodes can be created.
- `internalNetworkNames`: List of vSphere networks for connecting created nodes.
- `sshKeys`: Public SSH keys that will be added to the created virtual machines.

2. Apply the module configuration:

```
d8 k apply -f vsphere-mc.yaml
```

3. Wait for the `cloud-provider-vsphere` module to become ready:

```
d8 k get module cloud-provider-vsphere -o wide
```

4. Create a file with the VsphereInstanceClass and NodeGroup resources with the `nodeType: CloudEphemeral` value. For example, `vsphere-instance.yaml`:

```
apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: ephemeral
spec:
  numCPUs: 2
  memory: 4096
  rootDiskSize: 40
  template: "<PATH_TO_TEMPLATE_FROM_DATACENTER>"
  mainNetwork: "<PORT_GROUP_NAME>"
  datastore: "<DATASTORE_OR_FOLDER/DATASTORE>"
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ephemeral
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VsphereInstanceClass
      name: ephemeral
    maxPerZone: 1
    minPerZone: 1
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
  disruptions:
    approvalMode: Automatic
```

Where:

- VsphereInstanceClass describes the parameters of the virtual machine that will be created in vSphere.
- NodeGroup describes the node group that DP must maintain in the cluster.
- `nodeType: CloudEphemeral` means that nodes will be created automatically through the cloud provider.
- `cloudInstances.classReference` points to VsphereInstanceClass.
- `cloudInstances.zones` must contain zones from the `zones` list in ModuleConfig.

5. Apply the manifest:

   ```
   d8 k apply -f vsphere-instance.yaml
   ```

   After the manifest is applied, DP will start creating a virtual machine in vSphere. After the VM boots, kubelet will connect to the Kubernetes API, and the new node will appear in the cluster.

6. Check the node status:

   ```
   d8 k get nodes -o wide
   ```

   Example expected output:

   ```
   NAME                              STATUS   ROLES                  AGE   VERSION
   static-master-0                   Ready    control-plane,master   1h    v1.33.10
   ephemeral-1ca02a5b-7588b-k89dc    Ready    ephemeral              10m   v1.33.10
   ```

7. If VM creation fails, check the Machine and MachineSet objects and the machine-controller-manager logs:

   ```
   d8 k -n d8-cloud-instance-manager get machinesets,machines -o wide
   d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
   ```

   Also check the cluster events:

   ```
   d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
   ```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-vsphere` module is enabled:

  ```
  d8 k get module cloud-provider-vsphere -o wide
  ```

- The `cloud-provider-vsphere` module components are in the `Running` state:

```
d8 k -n d8-cloud-provider-vsphere get pods -o wide
```

- StorageClasses for vSphere have been created in the cluster:

```
d8 k get sc
```

- A virtual machine that will be connected to the cluster has been created in vSphere.
- The virtual machine name in vSphere matches the hostname inside the operating system.
- The following parameters are set in the VM advanced parameters in vSphere:

```
disk.EnableUUID = TRUE
guestinfo.metadata = <BASE64_ENCODED_METADATA>
guestinfo.metadata.encoding = base64
```

The `guestinfo.metadata` parameter must contain the metadata configuration encoded in Base64. Example `metadata.json` file:

```
{
   "instance-id": "cloud-static-worker-0",
   "local-hostname": "cloud-static-worker-0",
   "public-keys-data": "<SSH_PUBLIC_KEY>",
   "network": {
     "version": 2,
     "ethernets": {
       "id0": {
         "match": {
           "driver": "vmxnet3"
         },
         "set-name": "ens192",
         "dhcp4": true
       }
     }
   }
 }
```

Where:

- `instance-id`: Virtual machine identifier.
- `local-hostname`: Node hostname inside the operating system.
- `public-keys-data`: Public SSH key for accessing the virtual machine.
- `network`: Network settings that will be applied inside the virtual machine.

To get the value for the `guestinfo.metadata` parameter, run:

```
METADATA_B64="$(base64 -w0 metadata.json)"
echo "$METADATA_B64"
```

- The virtual machine is connected to the network specified in the `internalNetworkNames` parameter of the `cloud-provider-vsphere` module configuration.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a file with a NodeGroup resource and the CloudStatic node type. For example, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

2. Make sure that the NodeGroup has been created and synchronized:

```
d8 k get nodegroup cloud-static
```

Example expected output:

```
NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
STANDBY   STATUS   AGE   SYNCED
cloud-static   CloudStatic   0       0       0
1m    True
```

3. Get the bootstrap script for the created NodeGroup:

```
NODE_GROUP=cloud-static

d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
  -o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

4. Copy the bootstrap script to the virtual machine being connected:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

5. Connect to the virtual machine over SSH:

```
ssh <USER>@<NODE_IP>
```

6. On the virtual machine, set permissions and run the bootstrap script:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

```
d8 k get nodes -o wide
```

   Example expected output:

```
NAME                      STATUS   ROLES          AGE   VERSION    INTERNAL-IP
static-master-0           Ready    master         1h    v1.33.10   192.168.240.135
cloud-static-worker-0     Ready    cloud-static   5m    v1.33.10   192.168.240.152
```

Additional resources

- Global custom resources
- Module `cloud-provider-vsphere` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid cluster with built-in virtualization

This section describes how to add built-in virtualization nodes to an existing static DP cluster.

Integration with built-in virtualization uses the `cloud-provider-dvp` module. It enables DP to interact with the built-in virtualization cluster API, create virtual machines, connect the created VMs to an existing Kubernetes cluster, and manage the lifecycle of nodes through Cluster API mechanisms.

This section describes two ways to add nodes:

- **Automatically creating built-in virtualization nodes**. DP creates virtual machines through the built-in virtualization API. VM parameters are specified using the DVPInstanceClass resource, while the required number of nodes and placement zones are specified using the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes using a bootstrap script**. A virtual machine is created in advance by the user and connected to the cluster using a DP bootstrap script. This scenario uses a NodeGroup with the `CloudStatic` type.

Prerequisites

Before you begin, make sure that the following requirements are met:

- The cluster is created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of the static DP cluster nodes and the network of virtual machines in built-in virtualization. For details, see Network requirements. Nodes created in built-in virtualization have access to the Kubernetes API of the target DP cluster, DNS, and the required addresses according to the Network interaction and Configuring network policies sections. If Cilium is used with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites. The Kubernetes API of the built-in virtualization cluster is accessible from the DP cluster.
- The requirements from the Preparing the environment section are met:
  - a ServiceAccount has been created to access the built-in virtualization API;
  - a kubeconfig has been generated to connect to the built-in virtualization API;
  - a namespace has been prepared where virtual machines and disks will be created.
- A Linux OS image with `cloud-init` support is available in built-in virtualization, for example `ubuntu-24-04-lts`.
- A suitable VirtualMachineClass is available in virtualization, for example `amd-epyc-gen-3`.
- A StorageClass for root disks of virtual machines is available in built-in virtualization, for example `replicated`.
- If a virtual machine template is used, make sure that it contains only one disk.

DVPInstanceClass parameters use resources from the built-in virtualization cluster: VirtualMachineClass, ClusterVirtualImage, VirtualImage, VirtualDisk, and StorageClass from built-in virtualization, not from the target DP cluster.

Make sure there are no StorageClass resources in the static DP cluster whose names match the names of StorageClass resources in the built-in virtualization cluster.

When the `cloud-provider-dvp` module is enabled, the corresponding StorageClass resources are automatically synchronized from built-in virtualization to the DP cluster. If a StorageClass with the same name already exists in the static cluster, a resource conflict may occur, causing the module installation or upgrade to fail.

Adding automatically created nodes

1. On the administrator’s machine where access to the built-in virtualization cluster is configured, prepare a kubeconfig for the `cloud-provider-dvp` module to access the virtualization API.

   Follow the steps in the “Preparing the environment” section and encode the generated kubeconfig

in Base64:

```
export DVP_PROVIDER_KUBECONFIG="./kubeconfig"
export DVP_KUBECONFIG_B64="$(base64 -w0 ${DVP_PROVIDER_KUBECONFIG})"
```

2. Specify the built-in virtualization namespace where virtual machines and disks will be created:

   ```
   export DVP_NAMESPACE="<DVP_NAMESPACE>"
   ```

3. Specify the built-in virtualization zone where nodes will be created.

   Currently, zoning in built-in virtualization is under development, so use the `default` value for the `zones` parameters in ModuleConfig and NodeGroup:

   ```
   export DVP_ZONE="default"
   ```

   If necessary, you can check the topology labels of nodes in the built-in virtualization cluster:

   ```
   d8 k get nodes -L topology.kubernetes.io/region,topology.kubernetes.io/zone
   ```

   ⚠ The zone value in ModuleConfig and NodeGroup must match. Currently, only the `default` value is available in DVP.

4. Create a file with the `cloud-provider-dvp` module configuration. For example, `cloud-provider-dvp-mc.yaml`:

   ```
   cat > cloud-provider-dvp-mc.yaml <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cloud-provider-dvp
   spec:
     enabled: true
     version: 2
     settings:
       nodes:
         parameters:
           layout: Standard
           sshPublicKey: "<SSH_PUBLIC_KEY>"
           zones:
             - ${DVP_ZONE}
       provider:
         parameters:
           namespace: ${DVP_NAMESPACE}
   EOF
   ```

   The manifest uses the environment variables set in the previous steps: `DVP_NAMESPACE` and `DVP_ZONE`. Replace `<SSH_PUBLIC_KEY>` with the public SSH key for access to the created nodes.

5. Apply the manifest:

   ```
   d8 k apply -f cloud-provider-dvp-mc.yaml
   ```

6. Wait until the `d8-cloud-provider-dvp` namespace appears:

   ```
   d8 k get ns d8-cloud-provider-dvp
   ```

   The namespace is created by the `cloud-provider-dvp` module, so the credentials Secret is applied as a separate step rather than together with the ModuleConfig. Until the Secret exists, the module components cannot reach the DVP API.

7. Create the credentials Secret:

   ```
   d8 k apply -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: d8-credentials
     namespace: d8-cloud-provider-dvp
   type: cloud-provider.deckhouse.io/credentials
   stringData:
     authScheme: kubeconfig
     secret: ${DVP_KUBECONFIG_B64}
   EOF
   ```

   The Secret uses the `DVP_KUBECONFIG_B64` environment variable set in the first step.

8. Wait until the `cloud-provider-dvp` module is enabled:

   ```
   d8 k get module cloud-provider-dvp -o wide
   ```

   The module must switch to the `Ready` state, and the pods in the `d8-cloud-provider-dvp` namespace must be in the `Running` state.

9. Make sure that the `node-manager` module is in the `Ready` state:

   ```
   d8 k get module node-manager -o wide
   ```

   If the module is in the `Error` state, check that available built-in virtualization zones are specified in ModuleConfig and NodeGroup.

10. Make sure that the DVPInstanceClass resource is available in the cluster:

    ```
    d8 k get crd dvpinstanceclasses.deckhouse.io
    ```

11. On the administrator’s machine where access to the built-in virtualization cluster is configured, check the available virtual machine classes, images, and StorageClasses:

    ```
    d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get virtualmachineclasses
    d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get clustervirtualimages
    d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get storageclasses
    ```

    Use the obtained values when creating DVPInstanceClass.

12. Create a file with DVPInstanceClass and NodeGroup resources. For example, `dvp-instanceclass-nodegroup.yaml`:

```
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: dvp-worker
spec:
  virtualMachine:
    cpu:
      cores: 3
      coreFraction: 20%
    memory:
      size: 6Gi
    virtualMachineClassName: <VIRTUAL_MACHINE_CLASS_NAME>
    bootloader: EFI
  rootDisk:
    size: 15Gi
    storageClass: <STORAGE_CLASS_NAME>
    image:
      kind: ClusterVirtualImage
      name: <CLUSTER_VIRTUAL_IMAGE_NAME>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: dvp-worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: dvp-worker
    minPerZone: 1
    maxPerZone: 1
    zones:
      - default
```

Where:

- `virtualMachineClassName` — name of the VirtualMachineClass in built-in virtualization, for example `amd-epyc-gen-3`;
- `rootDisk.storageClass` — name of the StorageClass in built-in virtualization, for example `replicated`;
- `rootDisk.image.kind` — image source type. For a cluster image, use `ClusterVirtualImage`;
- `rootDisk.image.name` — name of the OS image in built-in virtualization, for example `ubuntu-24-04-lts`;
- `cloudInstances.zones` — built-in virtualization zone where the node will be created. The value must match the `zones` value in ModuleConfig.

13. Apply the manifest:

```
d8 k apply -f dvp-instanceclass-nodegroup.yaml
```

After the manifest is applied, DP will start creating a virtual machine in DVP and connect it to the cluster as a node.

14. Check the NodeGroup status:

```
d8 k get nodegroup dvp-worker -o wide
d8 k describe nodegroup dvp-worker
```

15. Check that a new node appears in the DP cluster:

```
d8 k get nodes -o wide
```

Example expected output:

```
NAME                              STATUS   ROLES                  AGE   VERSION
INTERNAL-IP
dvp-hybrid-master-0               Ready    control-plane,master   1h
v1.33.10   10.12.0.69
dvp-worker-c75a75c1-twqp4-bjpvl   Ready    dvp-worker             10m
v1.33.10   10.12.3.15
```

Adding manually created nodes using a bootstrap script

Before you begin, make sure that the following requirements are met:

- The `cloud-provider-dvp` module is enabled:

  ```
  d8 k get module cloud-provider-dvp -o wide
  ```

- The `cloud-provider-dvp` module components are in the `Running` state:

  ```
  d8 k -n d8-cloud-provider-dvp get pods -o wide
  ```

- A virtual machine that will be connected to the cluster has been created in built-in virtualization.
- The virtual machine is connected to the built-in virtualization network used for hybrid integration with the cluster.
- The virtual machine IP address belongs to the range specified in `internalNetworkCIDRs`.
- The virtual machine name in built-in virtualization matches the hostname inside the operating system.
- SSH access is available on the virtual machine for copying and running the bootstrap script.
- The SSH user can run commands using `sudo` without entering a password.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a NodeGroup with the `CloudStatic` node type. In this example and the following steps, the `cloud-static` name is used:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: cloud-static
   spec:
     nodeType: CloudStatic
   EOF
   ```

2. Make sure that NodeGroup has been created and synchronized:

   ```
   d8 k get nodegroup cloud-static
   ```

   Example expected output:

   ```
   NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED
   MIN   MAX   STANDBY   STATUS   AGE   SYNCED
   cloud-static   CloudStatic   0       0       0
   1m    True
   ```

3. Get the bootstrap script for the created NodeGroup:

   ```
   NODE_GROUP=cloud-static

   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
     -o jsonpath='{.data.bootstrap\.sh}' > ${NODE_GROUP}-bootstrap.b64
   ```

4. On the master node, verify that the file contains valid Base64 data of the bootstrap script:

```
base64 -d ${NODE_GROUP}-bootstrap.b64 > /dev/null
```

Check the beginning of the decoded content:

```
base64 -d ${NODE_GROUP}-bootstrap.b64 | head -n 5
```

The decoded content must start with a bash script:

```
#!/bin/bash
...
```

5. Copy the bootstrap script to the virtual machine being connected:

```
scp ${NODE_GROUP}-bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

6. Connect to the virtual machine over SSH:

```
ssh <USER>@<NODE_IP>
```

7. On the virtual machine, decode the bootstrap script, set permissions, and run it:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

After the bootstrap script starts, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

8. On the master node, check that a new node appears:

```
d8 k get nodes -o wide
```

Example expected output:

```
NAME                   STATUS   ROLES                  AGE   VERSION
INTERNAL-IP
dvp-hybrid-master-0    Ready    control-plane,master   1h    v1.33.12
10.12.0.69
cloud-static-worker-0  Ready    cloud-static           5m    v1.33.12
10.12.3.88
```

9. If connection fails, check the NodeGroup status, events, and bootstrap logs on the virtual machine being connected:

```
d8 k get nodegroup cloud-static
d8 k describe nodegroup cloud-static
d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
```

On the virtual machine being connected:

```
sudo tail -n 120 /var/log/d8/bashible/bootstrap.log
```

If the logs contain the `Failed to discover node_ip that matches internalNetworkCIDRs` error, check that the virtual machine IP address belongs to `internalNetworkCIDRs`.

Additional resources

- Global custom resources
- Module `cloud-provider-dvp` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

High reliability and availability

The reliability and resilience of a Kubernetes cluster are key characteristics that define the stability of the infrastructure. Deckhouse Platform (DP) ensures high availability (HA) and fault tolerance through built-in mechanisms and modules.

When HA mode is enabled, critical cluster components are launched with the necessary redundancy to guarantee their continuous operation. Even if a single instance fails, services can continue functioning without downtime. For more information on enabling HA mode, refer to Managing HA mode.

If the cluster has more than one master node, HA mode is automatically enabled, both during the initial deployment and when additional nodes are added later. Recommended roles and number of nodes can be found in Recommendations for configuring cluster nodes and preventing overload.

DP provides chaos engineering tools to test cluster resilience. These tools let you deliberately or randomly disrupt individual components and observe the infrastructure's response. For information on configuring these tools, refer to Chaos engineering.

You can further increase cluster fault tolerance by enabling inter-cluster communication based on the Service Mesh mode of the `istio` module. In this mode, federation is configured between multiple clusters. In case of failures in one cluster, the load is automatically redistributed to others. For configuration details, refer to Federation.

Virtual machines also survive node failures and overload. Rebalancing moves them off loaded nodes by live migration, and the ColdStandby mechanism brings machines back into operation after a node fails. For details, refer to Virtual machine fault tolerance and balancing.

Additional resources

- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Recommendations for configuring cluster nodes and preventing overload

Proper node configuration increases cluster fault tolerance and simplifies maintenance. Below are the recommended roles and number of nodes for various components.

Master nodes

To ensure cluster fault tolerance, **always** use at least three master nodes. This number ensures uninterrupted cluster operation and lets you update master nodes safely. More than three nodes are unnecessary, and two nodes are insufficient to maintain alignment (quorum) in case of issues in one of the nodes.

Using only one master node means its failure will bring down the entire cluster since it manages key components that keep the cluster running.

Frontend nodes

Frontend nodes balance incoming traffic. Ingress controllers run on them. Use more than one frontend node. These nodes must be able to handle traffic in case at least one frontend node fails.

For example, if the cluster has two frontend nodes, each must be capable of handling the full load if the other fails. If there are three, each node must be able to handle at least a 150% load increase.

Monitoring nodes

Monitoring nodes run Grafana, Prometheus, and other monitoring components. In heavily loaded clusters with multiple alerts and large metric volumes, it’s recommended that you allocate dedicated nodes for monitoring. Otherwise, monitoring components will run on system nodes.

When dedicating nodes for monitoring, it’s important they use fast disks (at least 400 IOPS).

System nodes

System nodes are used to run Deckhouse modules. Allocate two system nodes. This ensures Deckhouse modules will run on them without interfering with user applications in the cluster.

Preventing Kubernetes API overload (FlowSchema)

By default, a DP cluster includes a component that implements FlowSchema and PriorityLevelConfiguration to prevent Kubernetes API overload.

`FlowSchema` assigns `PriorityLevel` for `list` requests from all service accounts in Deckhouse namespaces (with the label `heritage: deckhouse`) to the following apiGroups:

- `v1` (Pod, Secret, ConfigMap, Node, etc.): Useful for clusters with numerous basic resources (for example, Secrets or Pods).
- `apps/v1` (DaemonSet, Deployment, StatefulSet, ReplicaSet, etc.): Useful when numerous applications are deployed in the cluster (for example, Deployments).
- `deckhouse.io` (Deckhouse custom resources): Useful when there are numerous Deckhouse custom resources in the cluster.
- `cilium.io` (Cilium custom resources): Useful for clusters with numerous Cilium policies.

All API requests matching a `FlowSchema` are placed into a single queue with a shared priority. This limits the load on the Kubernetes API and prevents it from being overloaded by mass `list` requests.

This component does not have settings, but the following commands are available:

- Check the state of priority levels:

  ```
  d8 k get --raw /debug/api_priority_and_fairness/dump_priority_levels
  ```

- Check the state of priority level queues:

  ```
  d8 k get --raw /debug/api_priority_and_fairness/dump_queues
  ```

The component also provides the following metrics to Grafana:

- `apiserver_flowcontrol_rejected_requests_total`: Total number of rejected requests.
- `apiserver_flowcontrol_dispatched_requests_total`: Total number of processed requests.
- `apiserver_flowcontrol_current_inqueue_requests`: Number of requests in queues.
- `apiserver_flowcontrol_current_executing_requests`: Number of requests being executed.

Delaying node reboot or shutdown while critical pods are running

DP supports a mechanism for delaying the shutdown or reboot of a node if it is running stateful applications or virtual machines (VMs). This is implemented by the `d8-shutdown-inhibitor` service of the `node-manager` module.

The delay is triggered if pods with the label `pod.deckhouse.io/inhibit-node-shutdown` are running on the node. DP delays the shutdown or restart of the node so that the user can migrate critical pods or terminate them in a controlled manner. This is useful, for example, for stateful workloads and other cases where data loss must be avoided. For information about the mechanism’s principles and features, refer to the `node-manager` module documentation.

Enabling a node shutdown delay

⚠ To decide whether to block a node shutdown, DP additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

To enable the mechanism that delays a node's restart or shutdown, add the label `pod.deckhouse.io/inhibit-node-shutdown` to the Pod (for a Deployment, specify the label in the pod template).

Example of a Pod manifest with the `pod.deckhouse.io/inhibit-node-shutdown` label:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```

⚠ The block only applies to pods labeled with `pod.deckhouse.io/inhibit-node-shutdown` that are in the `Running` phase on the node. Kubelet shuts down the pods without this label according to the standard graceful shutdown process.

Searching pods that prevent shutdown and performing the node troubleshooting

To search the pods that are blocking a node shutdown, use the following command:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -o wide
```

To view the `GracefulShutdownPostpone` condition on a node, use the following command:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?
(@.type=="GracefulShutdownPostpone")]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}
{.message}{"\n"}{end}'
```

Possible values of the `reason` field:

- `WaitingForShutdownSignal`: The shutdown blocking mechanism is running on the node, but the blocking has not yet begun.
- `PodsWithLabelAreRunningOnNode`: The shutdown has been blocked because there are pods with the label `pod.deckhouse.io/inhibit-node-shutdown` still running on the node.
- `NoRunningPodsWithLabel`: The blocking has been released and the shutdown can continue.

Additional resources

- Module `node-manager` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing HA mode

> **i** Note that if the cluster has **more than one master node**, HA mode is **enabled automatically**. This applies both when deploying a cluster with multiple master nodes from the start and when increasing the number of master nodes from one to three.

Enabling HA mode globally

You can enable HA mode globally for DP in one of the following ways.

Using ModuleConfig/global custom resource

1. Set the `settings.highAvailability` parameter to `true` in `ModuleConfig/global`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: global
   spec:
     version: 2
     settings:
       highAvailability: true
   ```

2. To ensure HA mode is enabled, you can, for example, check the number of `deckhouse` Pods in the `d8-system` namespace. To do that, run the following command:

   ```
   d8 k -n d8-system get po | grep deckhouse
   ```

   The number of `deckhouse` Pods in the output must be more than one:

   ```
   deckhouse-57695f4d68-8rk6l                          2/2     Running   0             3m49s
   deckhouse-5764gfud68-76dsb                          2/2     Running   0             3m49s
   deckhouse-fgrhy4536s-fhu6s                          2/2     Running   0             3m49s
   ```

Using Deckhouse web UI

If the `console` module is enabled in the cluster, open the Deckhouse web UI, navigate to **Deckhouse — Global settings — Global module settings**, and switch the **HA mode** toggle to **Yes**.

Configuring HA mode with two master nodes and an arbiter node

Deckhouse Platform allows you to configure HA mode with two master nodes and an arbiter node. This approach allows you to meet HA requirements in conditions of limited resources.

Only etcd is placed on the arbiter node, without the other control plane components. This node is used to ensure the etcd quorum.

Requirements for the arbiter node:

- At least 2 CPU cores
- At least 4 GB of RAM
- At least 8 GB of disk space for etcd

The network latency requirements for the arbiter node are similar to those for the master nodes.

Configuring in a cloud cluster

The example below applies to a cloud cluster with three master nodes. To configure HA mode with two master nodes and an arbiter node in a cloud cluster, you need to remove one master node from the cluster and add one arbiter node.

To do this, follow these steps:

> **⚠** The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2`.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. We strongly recommend creating a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (for example, to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DP queue is empty:

   ```
   d8 system queue list
   ```

5. On the **local machine**, run the DP installer container for the corresponding edition and version (change the container registry address if needed):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

6. **In the installer container**, run the following command:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-
   keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
     --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   Change the cloud provider settings:

   - Set `masterNodeGroup.replicas` to `2`.
   - Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

     ```
     nodeGroups:
       - name: arbiter
         replicas: 1
         nodeTemplate:
           labels:
             node.deckhouse.io/etcd-arbiter: ""
           taints:
             - key: node.deckhouse.io/etcd-arbiter
               effect: NoSchedule
         zones:
           - europe-west3-b
         instanceClass:
           machineType: n1-standard-4
       # ... the rest of the manifest
     ```

   - Save your changes.

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look as follows:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

7. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME>
   --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST>
   ```

   > **i** **Important**. For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in OpenStack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the OpenStack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the `kubernetes-data` disk remains, there may be problems with etcd operation as the number of master nodes increases.

8. Check the Deckhouse queue with the following command and make sure there are no errors:

   ```
   d8 system queue list
   ```

Configuring in a static cluster

To configure HA mode with two master nodes and an arbiter node in a static cluster, follow these steps:

1. Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

   ```
   apiVersion: deckhouse.io/v1
     kind: NodeGroup
     metadata:
       name: arbiter
     spec:
       nodeType: Static
       nodeTemplate:
         labels:
           node.deckhouse.io/etcd-arbiter: ""
         taints:
           - key: node.deckhouse.io/etcd-arbiter
             effect: NoSchedule
     # ... the rest of the manifest
   ```

2. Add a node to the cluster that will be used as an arbiter node in a way that is convenient for you.
3. Ensure that the added arbiter node is in the list of etcd cluster members.
4. Remove one master node from the cluster.

Enabling HA mode for individual components

Some DP modules may have their own HA mode settings. To enable HA mode in a specific module, set the `settings.highAvailability` parameter in its configuration. The HA mode operation in individual modules is independent of the global HA mode.

List of modules supporting individual HA mode:

- `deckhouse`
- `openvpn`
- `istio`
- `dashboard`
- `multitenancy-manager`
- `user-authn`
- `ingress-nginx`
- `prometheus-monitoring`
- `monitoring-kubernetes`
- `snapshot-controller`

To enable HA mode manually for a specific module, add the `settings.highAvailability` parameter to its configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    highAvailability: true
```

To ensure HA mode is enabled, check the number of Pods for the target module. For example, to verify the mode operation for the `deckhouse` module, check the number of corresponding Pods in the `d8-system` namespace by running the following command:

```
d8 k -n d8-system get po | grep deckhouse
```

The number of `deckhouse` Pods in the output must be more than one:

```
deckhouse-57695f4d68-8rk6l                    2/2     Running   0             3m49s
deckhouse-5764gfud68-76dsb                    2/2     Running   0             3m49s
deckhouse-fgrhy4536s-fhu6s                    2/2     Running   0             3m49s
```

Additional resources

- Global parameters
- Module `console` documentation
- Module `control-plane-manager` documentation
- Module `deckhouse` documentation
- Module `openvpn` documentation
- Module `stronghold` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Chaos engineering

> ⚠ Chaos engineering mode can only be enabled for node groups with `nodeType: CloudEphemeral`.

Enable chaos engineering mode for a node group in one of the following ways:

1. Define the `spec.chaos` parameter in the configuration of the target node group with two nested parameters:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

Where:

- `mode`: Operation mode. Two options are available:
  - `DrainAndDelete`: When triggered, drains the node and deletes it.
  - `Disabled`: Skips this specific node group.
- `period`: Chaos Monkey trigger interval. Defined as a string with hours and minutes: `30m`, `1h`, `2h30m`, `24h`.

Example configuration for a node group:

```
# NodeGroup for cloud nodes in AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  chaos:
    mode: DrainAndDelete
    period: 24h
```

2. If the `console` module is enabled in the cluster, open the Deckhouse web UI, go to the settings of the desired node group under **Nodes** — **Node Groups**, and enable Chaos Monkey in the **Chaos monkey settings** section by specifying the time intervals in the corresponding fields.

Additional resources

- Module `console` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual machine fault tolerance and balancing

Node load changes over time, and nodes sometimes fail. Deckhouse Platform (DP) balances the placement of virtual machines (VMs) across nodes and restarts machines from a failed node, while metrics help you find out why a machine runs slowly.

VM rebalancing

Over time, the distribution of virtual machines across nodes stops being even. The `descheduler` module restores the balance by moving VMs with live migration, without interrupting them. Enable this module, and the distribution is maintained without your involvement.

Rebalancing solves two tasks:

- It evens out the load. DP tracks how much CPU is reserved on each node and, when a node reserves more than 80%, moves some VMs to the nodes that reserve less than 50%.
- It restores correct placement. DP checks whether the current node meets the VM requirements and the rules of mutual VM placement. For example, if the rules forbid keeping certain VMs on the same node, the extra ones are moved.

DP sets the rebalancing parameters itself: with the `descheduler` module enabled, it creates a Descheduler resource named `virtualization` and selects in it only the machines that can be moved by live migration. How the eviction strategies work is covered in Pod eviction.

- Using the CLI
- Using the web interface

To view the rebalancing settings, run the following command:

```
d8 k get descheduler virtualization -o yaml
```

1. Go to the **System** tab, then to **Configuration → Deschedulers**.
2. Click **Create**.
3. In the **Create resource** window that opens, enter the resource name in the **Name** field.
4. On the **Configuration** tab, in the **Strategies** block, enable the strategies you need. The **Low node utilization (balancing)** strategy moves VMs from overloaded nodes, while **Inter-Pod Anti-Affinity violations** and **Node Affinity violations** restore correct placement.
5. Click **Apply**.

The created resources and the strategies enabled in them appear in the list of the section.

Rebalancing covers only the VMs that can leave their node by live migration. A VM that can’t be live migrated, for example one with a passed-through device, is never moved by rebalancing, because the only other way off the node is a restart. Such a VM is restarted only during node maintenance and only with the permission of an administrator.

Diagnosing a slow VM

A slowdown of a virtual machine has two different causes. The guest OS is either busy with its own computations, or waiting for the node to give it processor time. From the outside both cases look the same, as a loaded processor of the machine.

DP metrics tell them apart, so you don’t have to log in to the guest system. The “Virtualization VM Happiness” Grafana dashboard shows how long each machine waits for a processor and what it lacks, as well as which nodes are loaded more than the rest.

What to do next depends on which of the causes is confirmed.

- **The machine consumes its entire guaranteed processor share.** Moving it to another node doesn’t help, because the same share is guaranteed there. Raise the core fraction `.spec.cpu.coreFraction` or reduce the number of virtual cores, so that the guest OS doesn’t spread the load across the cores that get no processor time. With the `Auto` value, DP picks the share from actual consumption, so you can’t change it manually and have to change the number of cores or set an explicit percentage.
- **The machine waits for processor time without consuming its guaranteed share.** The node doesn’t deliver the declared guarantee, and moving the machine to a less loaded node eliminates the delays.

The guarantee isn’t absolute under CPU oversubscription. Processor time is distributed between machines in proportion to their shares, so a machine with a few loaded cores among many competing ones can get less than it’s guaranteed.

When freeing up a node, move the machine with the highest consumption rather than the one that has slowed down. Moving the main consumer frees the node resources, while moving the other machines has little effect.

Before a move, check whether migration is possible, so that freeing up a node doesn’t turn into a restart of a machine. A machine held by its storage is in many cases moved along with its disks, while a machine with a passed-through device can’t be moved by live migration at all.

Make sure as well that the cluster has a suitable node. The VirtualMachineClass limits the choice to the nodes whose processors match the class. If only the current node is left on that list, migration is impossible under any resource shortage, and what’s left is freeing up the node itself by moving the other machines off it, or assigning the machine a class with a wider choice of nodes.

DP measures storage and network latency by comparison with the rest of the cluster rather than against a fixed threshold, because the same disk latency is normal for a replicated volume and indicates a problem for a local one. If the storage is equally slow across the cluster, moving a machine doesn’t eliminate the delays.

ColdStandby

The ColdStandby mechanism returns a virtual machine to service after the failure of the node it was running on.

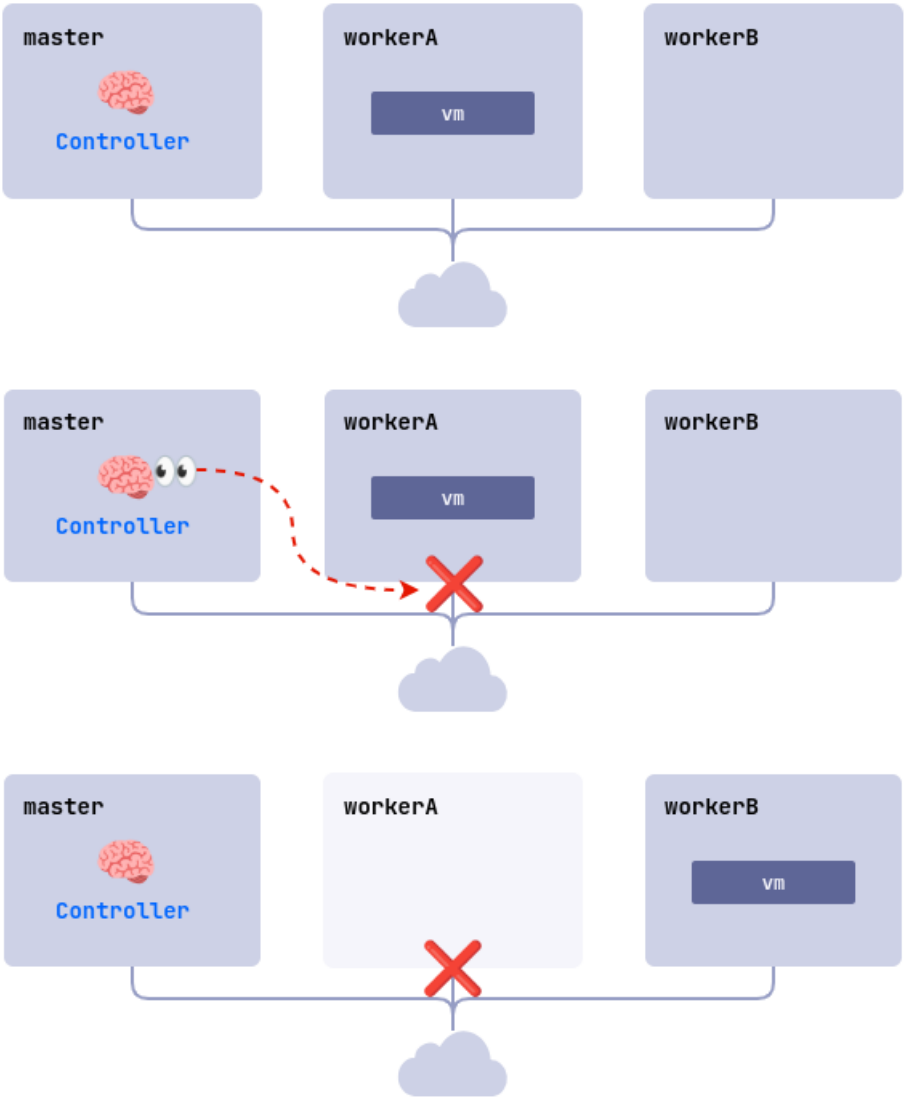
For the mechanism to work, meet two requirements:

- The `.spec.runPolicy` run policy of the virtual machine must be set to `AlwaysOnUnlessStoppedManually` or `AlwaysOn`.
- The Fencing mechanism must be enabled on the nodes that run virtual machines.

Without Fencing, the mechanism doesn’t work. An unavailable VM doesn’t move in that case, but stays on the failed node and resumes together with it.

Here’s the recovery sequence, using a cluster of three nodes, `master`, `workerA`, and `workerB`, where Fencing is enabled on both worker nodes and the `linux-vm` VM runs on `workerA`:

1. The `workerA` node fails, for example because of a power or network loss.
2. The controller checks node availability and finds that `workerA` doesn’t respond.
3. The controller removes `workerA` from the cluster.
4. The `linux-vm` VM starts on another suitable node, `workerB` in this example.



Additional resources

- Module `descheduler` documentation
- Module `node-manager` custom resources
- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Platform scaling and node management

Deckhouse Platform (DP) provides built-in mechanisms for comprehensive management of cluster architecture — both at the control plane level and at the node level.

Management capabilities include:

- Control plane management — automation of configuration and updates for Kubernetes API, etcd, and other system components; issuance and renewal of certificates; scaling of the control plane and transitioning between single-master and multi-master modes; high availability and recovery.
- Node management — creation and scaling of node groups using NodeGroup objects; automatic deployment and updating of nodes; support for various types of infrastructure (cloud and bare-metal); management of static and cloud nodes; use of additional tools for fine-tuning.

These capabilities allow the creation of reliable, scalable, and self-healing clusters, adapting them to any requirements for performance, high availability, and infrastructure constraints.

The following sections provide detailed descriptions of features, configuration examples, and best practices for effective control plane and node management in Deckhouse Platform.

If a cluster runs virtual machines, taking a node out for maintenance affects them as well. How to move machines to other nodes, what limits the migration, and what to do with machines that can’t be migrated is described in Maintenance of nodes running virtual machines.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

General management and configuration of the control plane

Main features

Deckhouse Platform (DP) manages control plane components using the `control-plane-manager` module, which runs on all master nodes (nodes with the label `node-role.kubernetes.io/control-plane: ""`).

The control plane management functionality includes:

- Managing certificates required for the control plane to function, including their renewal and issuance when the configuration changes. Secure configuration is maintained automatically, with the ability to quickly add additional SANs for secure access to the Kubernetes API.
- Component configuration. DP generates all necessary configurations and manifests (kube-apiserver, etcd, etc.), reducing the risk of human error.
- Upgrade/downgrade of components. DP supports consistent version upgrades or downgrades of the control plane, helping maintain version uniformity across the cluster.
- Managing the etcd cluster configuration and its members. DP scales master nodes and performs migrations between single-master and multi-master modes.
- Periodic defragmentation of etcd. In clusters with three or more etcd members, this feature is enabled by default.
- Configuring kubeconfig. DP generates an up-to-date configuration file (with `cluster-admin` permissions), creates, renews, and updates the kubeconfig for control-plane components and the admin kubeconfig (`admin.conf`), and creates a symlink for the `root` user (`/root/.kube/config` → `admin.conf`). When the `user-authz` module is enabled, the symlink can be disabled using the `rootKubeconfigSymlink` parameter in the `control-plane-manager` module (for more details, see the FAQ of the `control-plane-manager` module). DP also tightens access permissions for the `admin.conf` and `super-admin.conf` files to enhance security.

> **i** Some parameters affecting control plane behavior are taken from the ClusterConfiguration resource.

Enabling, disabling, and configuring the module

Enabling / disabling

You can enable or disable the `control-plane-manager` module in the following ways:

1. Create (or modify) the ModuleConfig/control-plane-manager resource by setting `spec.enabled` to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     enabled: true
   ```

2. Using the command:

   ```
   d8 system module enable control-plane-manager
   ```

   or to disable:

   ```
   d8 system module disable control-plane-manager
   ```

3. Via the Deckhouse web interface:

   - Go to the “Deckhouse → Modules” section.
   - Find the `control-plane-manager` module and click on it.
   - Toggle the “Module enabled” switch.

Configuration

To configure the module, use the ModuleConfig/control-plane-manager resource and specify the required parameters in `spec.settings`.

Example with the schema version, enabled module, and a few settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
      - bakery.infra
      - devs.infra
      publishAPI: {}
```

Checking DP status and queues

How to verify that `control-plane-manager` is running correctly and is not in a pending state, and how to check active DP tasks (queues):

1. Make sure the module is enabled:

   ```
   d8 k get modules control-plane-manager
   ```

2. Check the status of `control-plane-manager` pods (they run in the `kube-system` namespace and have the label `app=d8-control-plane-manager`):

   ```
   d8 k -n kube-system get pods -l app=d8-control-plane-manager -o wide
   ```

   Ensure that all pods are in the `Running` or `Completed` state.

3. Verify that master nodes are in the `Ready` state:

```
d8 k get nodes -l node-role.kubernetes.io/control-plane
```

To view detailed information:

```
d8 k describe node <node-name>
```

4. Get the list of queues and active tasks:

```
d8 system queue list
```

Example output:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```

⚠ Before performing heavy operations (e.g., transitioning from single-master to multi-master or upgrading the Kubernetes version), it is recommended to wait until all tasks in the queues are completed.

Certificate management

In DP, the `control-plane-manager` module is responsible for issuing and renewing all SSL certificates for control plane components. It manages:

1. **Server certificates** for kube-apiserver and etcd, stored in the `d8-pki` secret (namespace: `kube-system`):
   - Kubernetes root CA (`ca.crt`, `ca.key`).
   - etcd root CA (`etcd/ca.crt`, `etcd/ca.key`).
   - RSA certificate and key for signing Service Accounts (`sa.pub`, `sa.key`).
   - Root CA for extension API servers (`front-proxy-ca.key`, `front-proxy-ca.crt`).
2. **Client certificates** required for mutual communication between control plane components (e.g., `apiserver.crt`, `apiserver-etcd-client.crt`, etc.). These files are stored only on the nodes. If any changes occur (e.g., new SANs are added), certificates are automatically reissued, and the kubeconfig is synchronized.

PKI management

DP also manages the Public Key Infrastructure (PKI) used for encryption and authentication throughout the Kubernetes cluster:

- PKI for control plane components (kube-apiserver, kube-controller-manager, kube-scheduler, etc.).
- PKI for the etcd cluster (etcd certificates and inter-node communication).

DP assumes control of this PKI after the initial cluster installation and once its pods are running. As a result, all key issuance, renewal, and rotation operations (both for control plane and etcd) are performed automatically and centrally, without requiring manual intervention.

Additional SANs and auto-update

Deckhouse simplifies the addition of new Subject Alternative Names (SANs) for the Kubernetes API endpoint: you only need to specify them in the configuration. After any SAN change, the module automatically regenerates the certificates and updates the kubeconfig.

To add additional SANs (extra DNS names or IP addresses) for the Kubernetes API, specify the new SANs in the `spec.settings.apiserver.certSANs` field of your ModuleConfig/control-plane-manager resource.

DP will automatically generate new certificates and update all required configuration files (including kubeconfig).

Kubelet certificate rotation

In Deckhouse Platform, kubelet certificate rotation is automatic. The `--tls-cert-file` and `--tls-private-key-file` parameters for kubelet are not set directly. Instead, a dynamic TLS certificate

mechanism is used: kubelet applies the client certificate located at `/var/lib/kubelet/pki/kubelet-client-current.pem`, which it uses to request a new client or server certificate (file `/var/lib/kubelet/pki/kubelet-server-current.pem`) from kube-apiserver. Also, the CIS benchmark `AVD-KCV-0088` and `AVD-KCV-0089` checks, which track whether the `--tls-cert-file` and `--tls-private-key-file` arguments were passed to kubelet, are disabled in the `operator-trivy` module.

Features of kubelet certificate rotation in Deckhouse Platform:

- By default, kubelet generates its own keys in the `/var/lib/kubelet/pki/` directory and, if necessary, independently requests certificate renewal from kube-apiserver.
- lifetime of certificates is 1 year (8760 hours). When there are between 5 and 10% of the time remaining before expiration (the exact value is randomly selected from this range), kubelet automatically initiates a request for a new certificate. For more details, see the official documentation Kubernetes. If necessary, lifetime of certificates can be changed using `--cluster-signing-duration` parameter in `/etc/kubernetes/manifests/kube-controller-manager.yaml` manifest. However, in order for kubelet to obtain and install a new certificate before the current one expires, it is recommended to set the validity period of certificates to at least 1 hour.
- If the kubelet fails to renew the certificate before it expires, it will lose the ability to make requests to the kube-apiserver and, accordingly, renew certificates. As a result, the node will be marked as `NotReady` and automatically recreated.

Manual renewal of control plane certificates

If the master nodes were offline for an extended period (e.g., the servers were shut down), some control plane certificates may become outdated. In such cases, automatic renewal will not occur upon reboot — manual intervention is required.

To manually renew the control plane certificates, use the `kubeadm` utility on each master node:

1. Locate the `kubeadm` binary on the master node and create a symbolic link:

   ```
   ln -s  $(find /var/lib/containerd  -name kubeadm -type f -executable -print -quit) /usr/bin/kubeadm
   ```

2. Execute the following command:

   ```
   kubeadm certs renew all
   ```

   This command will regenerate the necessary certificates (for kube-apiserver, kube-controller-manager, kube-scheduler, etcd, and others).

Speeding up pod restarts after losing connection to a node

By default, if a node does not report its status within 40 seconds, it is marked as `Unreachable`. Then, after another 5 minutes, the pods on that node begin to be restarted on other nodes. As a result, the total application downtime can reach approximately 6 minutes.

In specific cases where an application cannot run in multiple instances, there is a way to reduce the downtime period:

1. Reduce the time before a node is marked as `Unreachable` by configuring the `nodeMonitorGracePeriodSeconds` parameter.
2. Set a shorter timeout for evicting pods from the unreachable node using the `failedNodePodEvictionTimeoutSeconds` parameter.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```

⚠ The shorter the timeouts, the more frequently system components check node status and plan pod rescheduling. This increases the load on the control plane, so choose values that match your requirements for high availability and performance.

Forcibly disabling IPv6 on cluster nodes

Internal communication between Deckhouse cluster components is performed via IPv4 protocol. However, at the operating system level of the cluster nodes, IPv6 is usually active by default. This leads to automatic assignment of IPv6 addresses to all network interfaces, including Pod interfaces. This results in unwanted network traffic - for example, redundant DNS queries like `AAAAA` - which can affect performance and make debugging network communications more difficult.

To correctly disable IPv6 at the node level in a Deckhouse-managed cluster, it is sufficient to set the necessary parameters via the NodeGroupConfiguration resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: disable-ipv6.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 50
  content: |
    GRUB_FILE_PATH="/etc/default/grub"

    if ! grep -q "ipv6.disable" "$GRUB_FILE_PATH"; then
      sed -E -e 's/^(GRUB_CMDLINE_LINUX_DEFAULT="[^"]*)"/\1 ipv6.disable=1"/' -i
"$GRUB_FILE_PATH"
      update-grub

      bb-flag-set reboot
    fi
```

⚠ After applying the resource, the GRUB settings will be updated and the cluster nodes will begin a sequential reboot to apply the changes.

etcd periodic defragmentation configuration

In clusters with three or more etcd members (3 and more master nodes or 2 and more master nodes if an etcd-arbiter is present), this feature is enabled by default. Defragmentation runs once a day at exactly 1:00 a.m. server time.

⚠ Enabling periodic defragmentation in a cluster with a single master node causes the control plane to be temporarily unavailable each time it runs.

To configure periodic etcd defragmentation, use the `etcd.defrag` parameter in the `control-plane-manager` module configuration. Specify the desired defragmentation frequency in the `etcd.defrag.cronSchedule` parameter using the cron (UTC) format.

Example configuration of the `control-plane-manager` module with settings for etcd defragmentation once a day at 4:00 a.m. server time:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
        - bakery.infra
        - devs.infra
      publishAPI: {}
    etcd:
      defrag:
        enabled: true
        cronSchedule: 0 4 * * *
```

Additional resources

- Global custom resources
- Module `console` documentation
- Module `control-plane-manager` documentation
- Module `node-manager` custom resources
- Module `operator-trivy` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Control plane recovery and debugging

Recovery from failures

During its operation DP automatically creates backups of configuration and data that may be useful in case of problems. These backups are saved in the `/etc/kubernetes/deckhouse/backup` directory. If any issues or unexpected situations occur during operation, you can use these backups to restore the system to a previously healthy state.

Restoring etcd cluster functionality

If the etcd cluster is not functioning and cannot be restored from a backup, you can attempt to recover it from scratch by following the steps below.

1. On all nodes that are part of your etcd cluster, **except one**, delete the `etcd.yaml` manifest located in `/etc/kubernetes/manifests/`. This will leave only one active node, from which the multi-master cluster state will be restored.
2. On the remaining node, open the `etcd.yaml` manifest and add the `--force-new-cluster` flag under `spec.containers.command`.
3. After the cluster is successfully restored, remove the `--force-new-cluster` flag.

> This operation is destructive: it completely wipes the existing data and initializes a new cluster based on the state preserved on the remaining node. All pending records will be lost.

Restoring a master node when kubelet fails to load control plane components

Such a situation may occur if images of the control plane components on the master were deleted in a cluster that has a single master node (e.g., the directory `/var/lib/containerd` was deleted). In this case, kubelet cannot pull images of the control plane components when restarted since the master node lacks authorization parameters required for accessing `registry.deckhouse.io`.

Below is an instruction on how you can restore the master node.

containerd

1. Execute the following command to restore the master node in any cluster running under DP:

   ```
   d8 k -n d8-system get secrets deckhouse-registry -o json |
   jq -r '.data.".dockerconfigjson"' | base64 -d |
   jq -r '.auths."registry.deckhouse.io".auth'
   ```

2. Copy the command's output and use it for setting the `AUTH` variable on the corrupted master.
3. Next, pull images of control plane components to the corrupted master:

   ```
   for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
     crictl pull --auth $AUTH $image
   done
   ```

4. Restart kubelet after pulling the images.

etcd restore

Viewing etcd cluster members

Option 1

Use the `etcdctl member list` command.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Warning. The last parameter in the output table shows etcd member is in `learner` state, is not in `leader` state.

Option 2

To obtain information about etcd cluster nodes in tabular form, use the `etcdctl endpoint status` command. For the leader, the `IS LEADER` column will show `true`.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Restoring the etcd cluster in case of complete unavailability

1. Stop all etcd nodes except one by deleting the `etcd.yaml` manifest on the others.
2. On the remaining node, add the `--force-new-cluster` option to the etcd startup command.
3. After the cluster is restored, remove this option.

✖ Be careful: these actions completely erase the previous data and form a new etcd cluster.

Recovering etcd after panic: unexpected removal of unknown remote peer error

1. Find the `etcdutl` utility on the master node and copy the executable to `/usr/local/bin/`:

   ```
   cp $(find /var/lib/containerd/ \
   -name etcdutl -print -quit) /usr/local/bin/etcdutl
   ```

2. Create a new etcd database snapshot from the current local snapshot (`/var/lib/etcd/member/snap/db`):

   ```
   etcdutl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
   --initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-
   urls=https://<ADDRESS>:2380 \
   --skip-hash-check=true --data-dir /var/lib/etcdtest
   ```

   where:

   - `<HOSTNAME>`: Name of the master node.
   - `<ADDRESS>`: Address of the master node.

3. Execute the following commands to use the new snapshot:

   ```
   cp -r /var/lib/etcd /tmp/etcd-backup
   rm -rf /var/lib/etcd
   mv /var/lib/etcdtest /var/lib/etcd
   ```

4. Locate the `etcd` and `api-server` containers:

   ```
   crictl ps -a | egrep "etcd|apiserver"
   ```

5. Remove the `etcd` and `api-server` containers:

   ```
   crictl rm <CONTAINER-ID>
   ```

6. Restart the master node.

Actions to take when etcd database exceeds quota-backend-bytes limit

When the database volume of etcd reaches the limit set by the `quota-backend-bytes` parameter, it switches to “read-only” mode. This means that the etcd database stops accepting new entries but remains available for reading data. You can tell that you are facing a similar situation by executing the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o
name | sed -n 1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ endpoint status -w table --cluster
```

If you see a message like `alarm:NOSPACE` in the `ERRORS` field, you need to take the following steps:

1. Make change to `/etc/kubernetes/manifests/etcd.yaml` — find the line with `--quota-backend-bytes` and increase the value by multiplying the specified number by two. If there is no such line — add, for example: `- --quota-backend-bytes=8589934592` — this sets the limit to 8 GB.

2. Disarm the active alarm that occurred due to reaching the limit. To do this, execute the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ alarm disarm
```

3. Change the `maxDbSize` parameter in the `control-plane-manager` settings to match the value specified in the manifest.

etcd defragmentation

i In clusters with three or more etcd members, periodic etcd defragmentation is enabled by default. It runs once a day at exactly 1:00 a.m. server time. For more details, see the `control-plane-manager` module documentation.

⚠ Before defragmenting, back up etcd.

To view the size of the etcd database on a specific node before and after defragmentation, use the command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  endpoint status --cluster -w table
```

Output example (the size of the etcd database on the node is specified in the `DB SIZE` column):

```
+----------------------------+-----------------+---------+----------------+------
---+--------+----------------------+--------+------------+------------+----------
+------------+--------------------+--------+--------------------------+-----------
------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB
SIZE | IN USE | PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM
| RAFT INDEX | RAFT APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE
ENABLED |
+----------------------------+-----------------+---------+----------------+------
---+--------+----------------------+--------+------------+------------+----------
+------------+--------------------+--------+--------------------------+-----------
------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76
MB |  62 MB |                   20% | 2.1 GB |       true |      false |        56 |
258054684 |          258054684 |        |                          |
false |
+----------------------------+-----------------+---------+----------------+------
---+--------+----------------------+--------+------------+------------+----------
+------------+--------------------+--------+--------------------------+-----------
------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76
MB |  62 MB |                   20% | 2.1 GB |      false |      false |        56 |
258054685 |          258054685 |        |                          |
false |
+----------------------------+-----------------+---------+----------------+------
---+--------+----------------------+--------+------------+------------+----------
+------------+--------------------+--------+--------------------------+-----------
------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76
MB |  62 MB |                   20% | 2.1 GB |      false |      false |        56 |
258054685 |          258054685 |        |                          |
false |
+----------------------------+-----------------+---------+----------------+------
---+--------+----------------------+--------+------------+------------+----------
+------------+--------------------+--------+--------------------------+-----------
------+
```

How to defragment an etcd node in a single-master cluster

⚠ Defragmenting etcd is a resource-intensive operation that temporarily blocks etcd from running on that node. Keep this in mind when choosing a time to perform the operation in a cluster with a single master node.

To defragment etcd in a cluster with a single master node, use the following command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

i If a timeout error occurs, increase the value of the `—command-timeout` parameter from the command above until defragmentation is successful.

How to defragment etcd in a cluster with multiple master nodes

To defragment etcd in a cluster with multiple master nodes:

1. Get a list of etcd pods. To do this, use the following command:

   ```
   d8 k -n kube-system get pod -l component=etcd -o wide
   ```

   Example output:

   ```
   NAME            READY    STATUS    RESTARTS   AGE     IP               NODE
   NOMINATED NODE   READINESS GATES
   etcd-master-0   1/1     Running   0          3d21h   192.168.199.80   master-0
   <none>           <none>
   etcd-master-1   1/1     Running   0          3d21h   192.168.199.81   master-1
   <none>           <none>
   etcd-master-2   1/1     Running   0          3d21h   192.168.199.82   master-2
   <none>           <none>
   ```

2. Identify the leader master node. To do this, contact any etcd pod and get a list of nodes participating in the etcd cluster using the command (where `NODE_NAME` is the name of the master node):

   ```
   d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
     --cert=/etc/kubernetes/pki/etcd/server.crt \
     --key=/etc/kubernetes/pki/etcd/server.key \
     --cacert=/etc/kubernetes/pki/etcd/ca.crt \
     endpoint status --cluster -w table
   ```

   Output example (the leader in the `IS LEADER` column will have the value `true`):

```
+----------------------------+------------------+---------+-----------------+-
--------+--------+----------------------+--------+------------+------------+--
---------+------------+--------------------+--------+--------------------------
+-------------------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION |
DB SIZE | IN USE | PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION
| DOWNGRADE ENABLED |
+----------------------------+------------------+---------+-----------------+-
--------+--------+----------------------+--------+------------+------------+--
---------+------------+--------------------+--------+--------------------------
+-------------------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |
76 MB |  62 MB |                  20% | 2.1 GB |       true |      false |
56 |  258054684 |          258054684 |        |                            |
false |
+----------------------------+------------------+---------+-----------------+-
--------+--------+----------------------+--------+------------+------------+--
---------+------------+--------------------+--------+--------------------------
+-------------------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |
76 MB |  62 MB |                  20% | 2.1 GB |      false |      false |
56 |  258054685 |          258054685 |        |                            |
false |
+----------------------------+------------------+---------+-----------------+-
--------+--------+----------------------+--------+------------+------------+--
---------+------------+--------------------+--------+--------------------------
+-------------------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |
76 MB |  62 MB |                  20% | 2.1 GB |      false |      false |
56 |  258054685 |          258054685 |        |                            |
false |
+----------------------------+------------------+---------+-----------------+-
--------+--------+----------------------+--------+------------+------------+--
---------+------------+--------------------+--------+--------------------------
+-------------------+
```

3. Defragment the etcd nodes that are members of the etcd cluster one by one. Use the following command to defragment (where `NODE_NAME` is the name of the master node):

> **i** Important: Defragment the leader last.
>
> Restoring etcd on a node after defragmentation may take some time. It is recommended to wait at least a minute before proceeding to defragment the next etcd node.

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `–command-timeout` parameter from the command above until defragmentation is successful.

High availability

If any component of the control plane becomes unavailable, the cluster temporarily maintains its current state but cannot process new events. For example:

- If `kube-controller-manager` fails, Deployment scaling will stop working.
- If `kube-apiserver` is unavailable, no requests can be made to the Kubernetes API, although existing applications will continue to function.

However, prolonged unavailability of control plane components disrupts the processing of new objects, handling of node failures, and other operations. Over time, this can lead to cluster degradation and impact user applications.

To mitigate these risks, the control plane should be scaled to a high-availability configuration — a minimum of three nodes. This is especially critical for etcd, which requires a quorum to elect a leader. The quorum works on a majority basis (N/2 + 1) of the total number of nodes.

Example:

Cluster size	Quorum (majority)	Max fault tolerance
1	1	0
3	2	1
5	3	2
7	4	3
9	5	4

> **i** An even number of nodes does not improve fault tolerance but increases replication overhead.

In most cases, three etcd nodes are sufficient. Use five if high availability is critical. More than seven is rarely necessary and not recommended due to high resource consumption.

After new control plane nodes are added:

- The label `node-role.kubernetes.io/control-plane=""` is applied.
- A DaemonSet launches control plane pods on the new nodes.
- DP creates or updates files in `/etc/kubernetes`: manifests, configuration files, certificates, etc.
- All DP modules that support high availability will enable it automatically, unless the global setting `highAvailability` is manually overridden.

Control plane node removal is performed in reverse:

- Labels `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master`, and `node.deckhouse.io/group` are removed.
- DP removes its pods from these nodes.
- etcd members on the nodes are automatically deleted.
- If the number of nodes drops from two to one, etcd may enter `readonly` mode. In this case, you must start etcd with the `--force-new-cluster` flag, which should be removed after a successful startup.

Additional resources

- Global parameters
- Module `control-plane-manager` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Scaling and changing master nodes

Scaling and single/multi-master transition

Control plane operation modes

Deckhouse Platform (DP) supports two operation modes for the control plane:

1. **Single-master**:
   - `kube-apiserver` uses only the local `etcd` instance.
   - A proxy server runs on the node to handle requests on `localhost`.
   - The `kube-apiserver` listens only on the master node's IP address.
2. **Multi-master**:
   - `kube-apiserver` interacts with all `etcd` instances in the cluster.
   - A proxy is configured on all nodes:
     - If the local `kube-apiserver` is unavailable, requests are automatically forwarded to other nodes.
   - This ensures high availability and supports scaling.

Automatic scaling of master nodes

DP allows automatic addition and removal of master nodes using the label `node-role.kubernetes.io/control-plane=""`.

Automatic control of master nodes includes:

- **Adding the label** `node-role.kubernetes.io/control-plane=""` to a node:
  - All control plane components are deployed.
  - The node is added to the etcd cluster.
  - Certificates and configuration files are regenerated automatically.
- **Removing the label**:
  - Control plane components are removed.
  - The node is properly removed from the etcd cluster.
  - Related configuration files are updated.

> **i** Transitioning from 2 to 1 master node requires manual etcd adjustment. All other changes in master node count are handled automatically.

Common scaling scenarios

> **⚠** For cluster stability, it is necessary to maintain an odd number of nodes with etcd to ensure quorum.

DP supports both automatic and manual scaling of master nodes in cloud and bare-metal clusters:

1. **Single-master → Multi-master**:
   - Add one or more master nodes.
   - Apply the label `node-role.kubernetes.io/control-plane=""` to them.
   - DP will automatically:
     - Deploy all control plane components.
     - Configure the nodes to work with the `etcd` cluster.
     - Synchronize certificates and configuration files.
2. **Multi-master → Single-master**:
   - Remove the labels `node-role.kubernetes.io/control-plane=""` and `node-role.kubernetes.io/master=""` from the extra master nodes.
   - For **bare-metal clusters**:
     - To correctly remove the nodes from `etcd`:
       - Run `d8 k delete node <node-name>`;
       - Power off the corresponding VMs or servers.

   > **i** **Important**. In cloud clusters, all necessary actions are automatically handled by the `dhctl converge` command.

3. **Changing the number of master nodes in a cloud cluster**:
   - Similar to node addition/removal, typically done using the `dhctl converge` command or cloud tools.
4. **Multi-master migration (3 master nodes) → 2 master nodes and 1 arbiter node in a**

4. **Multi-master migration (3 master nodes) → 2 master nodes and 1 arbiter node in a cloud cluster**:

   - In the cloud provider settings, set `masterNodeGroup.replicas` to `2` and create a NodeGroup for the arbiter node. Similar to node addition/removal, typically done using the `dhctl converge` command or cloud tools.

     For details on configuring HA mode with 2 master nodes and 1 arbiter node in a cloud cluster, refer to Managing HA mode.

5. **Multi-master migration (3 master nodes) → 2 master nodes and 1 arbiter node in a static cluster**:

   - Create a NodeGroup for the arbiter node and add the node to the cluster.
   - Remove the labels `node-role.kubernetes.io/control-plane=""`, `node-role.kubernetes.io/master=""` and `node.deckhouse.io/group-""` from the extra master node.
   - To correctly remove the node from etcd in **bare-metal clusters**:
     - Run `d8 k delete node <node-name>`.
     - Power off the corresponding VM or server.

     For details on configuring HA mode with 2 master nodes and 1 arbiter node in a static cluster, refer to Managing HA mode.

Removing the master role from a node without deleting the node itself

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

If you need to remove a node from the set of master nodes but keep it in the cluster for other purposes, follow these steps:

1. Remove the labels so the node is no longer treated as a master:

```
d8 k label node <node-name> node-role.kubernetes.io/control-plane-
d8 k label node <node-name> node-role.kubernetes.io/master-
d8 k label node <node-name> node.deckhouse.io/group-
```

2. Make sure that the master node to be deleted is no longer listed as a member of the etcd cluster:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

3. Delete the static manifests of the control plane components so they no longer start on the node, and remove unnecessary PKI files. Exec to the node and run the following commands:

```
rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
rm -f /etc/kubernetes/authorization-webhook-config.yaml
rm -f /etc/kubernetes/admin.conf /root/.kube/config
rm -rf /etc/kubernetes/deckhouse
rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
rm -rf /var/lib/etcd/member/
```

After completing these steps, the node will no longer be considered a master node, but it will remain part of the cluster and can be used for other tasks.

Changing the OS image of master nodes in a multi-master cluster

The OS change method depends on the cluster type: in a cloud cluster, nodes are replaced with `dhctl converge`; in a static cluster, nodes are replaced manually, one at a time.

For moving the nodes of the other types to a new image, see “How do I replace the OS image in a cloud cluster?”.

In a cloud cluster

To change the OS image of master nodes in a cloud multi-master cluster, follow these steps:

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module’s data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Make sure there are no alerts in the cluster that could interfere with updating master nodes.
4. Ensure the DP queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. **On your local machine**, run the Deckhouse installer container for the corresponding edition and version (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command to check the state before starting the operation:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
   user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
   HOST>
   ```

   The output should indicate that Terraform has found no discrepancies and no changes are required.

8. **In the installer container**, run the following command and specify the desired OS image in the `masterNodeGroup.instanceClass` parameter
   (provide all master node addresses using the `--ssh-host` parameter):

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-
   keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
   HOST>
   ```

9. **In the installer container**, run the following command to update the nodes:

   Carefully review the actions that `converge` plans to perform when it prompts for confirmation.

During execution, nodes will be replaced with new ones, one by one, starting from the highest numbered node (2) down to the lowest (0):

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME>
\
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
HOST>
```

The following steps should be performed **sequentially on each** master node, starting with the highest numbered node (with suffix 2) and ending with the lowest (with suffix 0).

10. **On the newly created node**, open the systemd journal for the `bashible.service`.
    Wait until the setup process is complete — the log should contain the message `nothing to do`:

    ```
    journalctl -fu bashible.service
    ```

11. Verify that the etcd node appears in the cluster node list:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

12. Make sure that `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
    ```

13. Check the Deckhouse queue and make sure that there are no errors with the command:

    ```
    d8 system queue list
    ```

14. Proceed to updating the next master node.

In a static cluster

The following instruction describes OS replacement on master nodes that were added to the cluster **manually** using the `bootstrap.sh` script. Perform the steps for each master node, one node at a time. Do not proceed to the next node until the current node has rejoined the cluster and is healthy.

⚠ If master nodes are managed by Cluster API Provider Static (CAPS) through StaticInstance resources, do not use this instruction. First delete the StaticInstance, install the required OS, and add the node to the `master` NodeGroup again.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

To change the OS of a manually added master node, follow these steps:

1. Create a backup of etcd and the `/etc/kubernetes` directory. If the `stronghold` module is enabled, make sure its data has also been backed up.
2. Check the cluster health and make sure there are no alerts or pending Deckhouse queue tasks:

   ```
   d8 status
   ```

3. Remove the master node labels from the node:

```
d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
  node-role.kubernetes.io/master- node.deckhouse.io/group-
```

where `<MASTER_NODE_NAME>`: Name of the master node being changed.

4. Make sure the node has been removed from the etcd cluster member list:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

5. Evict workloads from the node with `d8 k drain`:

```
d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
```

6. Force-delete any Pods remaining on the node:

```
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --
force
```

7. Delete the Node object:

```
d8 k delete node <MASTER_NODE_NAME>
```

8. Clean up the DP data on the removed master node. This command irreversibly removes Kubernetes and DP data from the node. Before running it, make sure you have selected the correct node and created the required backups:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-
doing
```

9. Install the required OS on the node.
10. On one of the remaining master nodes in the cluster, retrieve and decode the script for adding a master node:

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
  -o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
```

11. Securely copy the `bootstrap.sh` file obtained in the previous step to the node being added and run it as `root` on that node:

```
bash bootstrap.sh
```

12. **On the node being added**, open the systemd journal for the `bashible.service`. Wait until the setup process is complete — the log should contain the message `nothing to do`:

```
journalctl -fu bashible.service
```

13. Wait until the node reaches the `Ready` status:

```
d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
```

14. Make sure the node is listed as an etcd cluster member:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

15. Make sure that `control-plane-manager` is running on the node:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
  -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
```

16. Make sure there are no alerts or pending queue tasks:

```
d8 status
```

17. Repeat the procedure for the next master node.

Changing the OS image in a single-master cluster

The method depends on the cluster type: first add extra master nodes, replace the OS in multi-master mode, then restore the original number of master nodes.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

In a cloud cluster

1. Convert the single-master cluster into a multi-master one according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Convert the multi-master cluster back to a single-master one following the instructions.

In a static cluster

1. Add extra master nodes according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Remove extra master nodes from the control plane role according to the instructions. Then delete them from the cluster with `d8 k delete node <MASTER_NODE_NAME>` and power off the corresponding servers.

Adding master nodes to a static or hybrid cluster

i It is important to have an odd number of masters to ensure a quorum.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

When installing Deckhouse Platform with default settings, the NodeGroup `master` lacks the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Because of this, after changing the number of `staticInstances` nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), when adding a regular node using Cluster API Provider Static (CAPS), it can be "intercepted" and added to the NodeGroup `master`, even if the corresponding `StaticInstance` (in `metadata`) specifies a label with a `role` different from `master`. To avoid this "interception", after installing DP, edit the NodeGroup `master` — add the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Example of NodeGroup `master` with `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Next, when adding master nodes to the cluster using CAPS, specify the label specified in `spec.staticInstances.labelSelector` NodeGroup `master` in the corresponding `StaticInstance`. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # The label specified in spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

i

When adding new master nodes using CAPS and changing the number of master nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), please note the following:

During the initial cluster installation, the configuration specifies the first master node on which the installation takes place. If, after installation, you need to create a multi-master cluster and add master nodes using CAPS, you must specify the number of nodes in the `spec.staticInstances.count` parameter of the NodeGroup `master` as one less than the desired number.

For example, if you need to create a multi-master with three master nodes in `spec.staticInstances.count` NodeGroup `master`, specify the value `2` and create two `staticInstances` for the nodes to be added. After adding them to the cluster, the number of master nodes will be three: the master node on which the installation took place and two master nodes added using CAPS.

Otherwise, adding a master node to a static or hybrid cluster is similar to adding a regular node. To do this, use the corresponding examples. All the necessary actions to configure a cluster control plane components on the new master nodes are performed automatically. Wait until the master nodes appear in `Ready` status.

Adding master nodes in a cloud cluster

This section describes how to convert a single-master cluster into a multi-master cluster.

⚠ Before adding nodes, make sure the required quotas are available. It’s important to have an odd number of master nodes to maintain etcd quorum.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module’s data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no active alerts in the cluster that may interfere with adding new master nodes.
4. Make sure the Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   > **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. On the **local machine**, run the Deckhouse installer container for the appropriate edition and version (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command to verify the state before proceeding:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
   user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   The output should confirm that Terraform found no differences and no changes are needed.

8. **In the installer container**, run the following command and set the target number of master nodes in the `masterNodeGroup.replicas` parameter:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-
   keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST>
   ```

   > **i** For **Yandex Cloud**, if public IPs are assigned to master nodes, the number of elements in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using the `Auto` value (automatic public IP assignment), the number of items in the array must still match.
   >
   > For example, with three master nodes (`masterNodeGroup.replicas: 3`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
   >
   > ```
   > externalIPAddresses:
   > - "Auto"
   > - "Auto"
   > - "Auto"
   > ```

9. **In the installer container**, run the following command to trigger scaling:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME>
--ssh-host <MASTER-NODE-0-HOST>
```

10. Wait until the required number of master nodes reaches the `Ready` status and all `control-plane-manager` pods become ready:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-
control-plane-manager
```

Reducing the number of master nodes in a cloud cluster

This section describes the process of converting a multi-master cluster into a single-master cluster.

⚠ The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2`.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DP queue is empty:

```
d8 system queue list
```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

```
docker login registry.deckhouse.io
```

During the login process, you will need to enter your `Username` and `Password`.

i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. On the **local machine**, run the DP installer container for the corresponding edition and version (change the container registry address if needed):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
'[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command and set `masterNodeGroup.replicas` to `1`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-
keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
  --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

8. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME>
   --ssh-host <MASTER-NODE-0-HOST>
   ```

   > **i** For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in Openstack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the Openstack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the kubernetes-data disk remains, there may be problems with ETCD operation as the number of master nodes increases.

9. Check the Deckhouse queue and make sure that there are no errors with the command:

   ```
   d8 system queue list
   ```

Accessing the DP controller in a multi-master cluster

In clusters with multiple master nodes, DP runs in high-availability mode (with multiple replicas). To access the active DP controller, you can use the following command (example shown for the `deckhouse-controller queue list` command):

```
d8 system queue list
```

Additional resources

- Module `control-plane-manager` documentation
- Module `node-manager` custom resources
- Module `stronghold` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Node management basics in Deckhouse

Deckhouse Platform (DP) supports the full lifecycle of node management:

- Automatic node scaling based on workload.
- Node updates and maintenance to keep them up to date.
- Centralized configuration management for node groups using the NodeGroup CRD.
- Support for various types of nodes: permanent, ephemeral, cloud-based, or bare-metal.

> **i** DP can operate on both bare-metal and cloud-based clusters, providing flexibility and scalability.

Node groups allow logical segmentation of the cluster infrastructure. In DP, the following NodeGroup roles are commonly used:

- `master`: Control plane nodes.
- `front`: Nodes for routing HTTP(S) traffic.
- `monitoring`: Nodes for hosting monitoring components.
- `worker`: Nodes for user applications.
- `system`: Dedicated nodes for system components.

Each group can have centralized configuration settings, including the Kubernetes version, resources, taints, labels, kubelet parameters, and more.

> **i** Node management is only available for clusters fully managed by DP. This feature is not supported for Managed Kubernetes clusters.

Automatic deployment and updates

Deckhouse Platform (DP) provides an automated mechanism for managing the lifecycle of nodes based on NodeGroup resources. DP supports both initial node provisioning and updates when configuration changes, for both cloud and bare-metal clusters (if the `node-manager` module is enabled).

How it works:

1. A NodeGroup is the primary resource for managing groups of nodes. It defines the node type, number of nodes, resource templates, and key parameters (e.g., kubelet settings, taints, etc.).
2. When a NodeGroup is created or modified, the `node-manager` module automatically reconciles the node state with the specified configuration.
3. Updates occur without user intervention — outdated nodes are removed, and new ones are created automatically.

Let’s take a look at automatic updates using the example of a kubelet version upgrade.

1. The user updates the `kubelet` section in the NodeGroup specification.
2. DP detects that current nodes do not match the new configuration.
3. New nodes with the updated settings are created sequentially.
4. Old nodes are gradually removed from the cluster.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-cloud
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: AnotherCloudInstanceClass
      name: my-class
```

Basic node and OS configuration

When nodes are created and joined to the cluster, DP automatically performs a series of actions required for proper cluster operation:

- Installing and configuring a supported operating system.
- Disabling automatic package updates.
- Setting up logging and system parameters.
- Installing necessary packages and utilities.
- Configuring the `kubernetes-api-proxy` component to balance traffic between `kubelet` and API servers.
- Installing and configuring the container runtime (`containerd`) and `kubelet`.
- Registering the node with the Kubernetes cluster.

These actions are performed automatically when using `bootstrap.sh` or when connecting nodes via StaticInstance and SSHCredentials resources.

Disruptive updates

Some updates — for example, upgrading `containerd` or kubelet across multiple versions — require node downtime and may cause short-term disruption of system components (*disruptive updates*).
The application mode for such updates is configured via the `disruptions.approvalMode` parameter:

- `Manual`: Manual approval mode for disruptive updates.
  When a disruptive update is available, the `NodeRequiresDisruptionApprovalForUpdate` alert is triggered.

  To approve the update, add the annotation `update.node.deckhouse.io/disruption-approved=` to each node in the group. Example:

  ```
  d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=
  ```

  > **Important**: In this mode, the node is not drained automatically.
  > If needed, perform the drain manually before applying the annotation.
  >
  > To avoid issues during draining, always use the `Manual` mode for master node groups.

- `Automatic`: Automatic approval mode for disruptive updates.

  In this mode, the node is drained automatically before applying the update by default.
  This behavior can be changed using the `disruptions.automatic.drainBeforeApproval` parameter in the node group settings.

- `RollingUpdate`: A mode in which a new node with updated settings is created and the old one is removed.
  Applicable only to cloud nodes.

  In this mode, an additional node is created in the cluster during the update.
  This can be useful if the cluster lacks sufficient resources to temporarily relocate workloads from the updating node.

Example of a system NodeGroup

System nodes are dedicated to running system components.
They are typically isolated using labels and taints to prevent user pods from being scheduled on them.
System nodes can be either static or cloud-based.

Example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  nodeType: Static
```

Example NodeGroupConfiguration descriptions

Installing the cert-manager plugin for kubectl on master nodes

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-
manager/releases/download/v1.7.1/kubectl-cert_manager-linux-amd64.tar.gz -o - | tar
-zxvf - kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Werf configuration for ignoring the Ready status of a node group

Werf checks the `Ready` status of resources and, if available, waits for the value to become `True`.

Creating (or updating) a NodeGroup resource in the cluster may take a significant amount of time (until all nodes become ready). When using werf (e.g., in CI/CD), this can lead to a build timeout.

To make werf ignore the NodeGroup status, add the following annotations to the NodeGroup resource:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

Settings for Static and CloudStatic NodeGroups

Node groups with types Static and CloudStatic are intended for managing manually created nodes — either physical (bare-metal) or virtual (in the cloud, but outside DP automation). These nodes are connected manually or via StaticInstance and do not support automatic updates or scaling.

Configuration specifics:

- All update operations (e.g., kubelet updates, node restarts, replacements) must be performed manually or through external automation tools outside of DP.
- It is recommended to explicitly set the desired kubelet version to ensure consistency across nodes, especially if they are added with different versions manually:

  ```
  nodeTemplate:
    kubelet:
      version: "1.28"
  ```

- Node registration to the cluster can be performed either manually or automatically, depending on the configuration:
  - **Manual**: The user downloads the bootstrap script, configures the server, and runs the script manually.
  - **Automatic (CAPS)**: When using StaticInstance and SSHCredentials, DP automatically connects and configures the nodes.
  - **Hybrid approach**: A manually added node can be handed over to CAPS by using the annotation `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

If the Cluster API Provider Static (CAPS) is enabled, the NodeGroup resource can use the `staticInstances` section. This allows DP to automatically connect, configure, and, if necessary, clean up static nodes based on StaticInstance and SSHCredentials resources.

> **i** In a NodeGroup of type Static or CloudStatic, you can explicitly specify the number of nodes using the `spec.staticInstances.count` parameter. This allows you to define the expected number of nodes — DP uses this value for state monitoring and automation.

Running DP on an arbitrary node

To run DP on an arbitrary node, configure the `deckhouse` module with the appropriate `nodeSelector` parameter and **do not** specify `tolerations`. The required `tolerations` will be set automatically in this case.

> **⚠** Only use nodes of type **CloudStatic** or **Static** to run DP. Avoid using a `NodeGroup` that contains only a single node for running DP.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    nodeSelector:
      node-role.deckhouse.io/deckhouse: ""
```

Additional resources

- Module `deckhouse` parameters
- Module `node-manager` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Adding and managing cloud nodes

In Deckhouse Platform (DP), cloud nodes can be of the following types:

- **CloudEphemeral** — temporary nodes that are automatically created and deleted.
- **CloudPermanent** — permanent nodes managed manually via `replicas`.
- **CloudStatic** — static cloud nodes. The machines are created manually or by external tools and DP connects them to the cluster and manages them just like the regular nodes.

Below are instructions for adding and configuring each type.

Adding CloudEphemeral nodes in a cloud cluster

CloudEphemeral nodes are automatically created and managed within the cluster using the Machine Controller Manager (MCM) or Cluster API (depending on configuration) — both components are part of the `node-manager` module in DP.

To add nodes:

1. Ensure that the cloud provider module is enabled. For example: `cloud-provider-yandex`, `cloud-provider-openstack`, `cloud-provider-aws`.
2. Create an InstanceClass object with the machine configuration. This object describes the parameters of the virtual machines to be created in the cloud.

   Example (for OpenStack):

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: worker-instance
spec:
  flavorName: m1.medium
  imageName: ubuntu-24-04-cloud-amd64
  rootDiskSize: 20
  mainNetwork: default
```

The following parameters are specified:

- `flavorName`: Instance type (CPU/RAM).
- `imageName`: OS image.
- `rootDiskSize`: Size of the root disk.
- `mainNetwork`: Cloud network for the instance.

3. Create a NodeGroup with the `CloudEphemeral` type. Example manifest:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: worker-instance
    minPerZone: 1
    maxPerZone: 3
    zones:
      - nova
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
    taints: []
```

4. Wait for the nodes to be launched and added automatically.

Configuration for CloudEphemeral NodeGroups

NodeGroups of CloudEphemeral type are designed for automatic scaling by creating and removing virtual machines in the cloud using the Machine Controller Manager (MCM). This type of groups is commonly used in cloud-based DP clusters.

Node configuration is defined in the `cloudInstances` section and includes parameters for scaling, zoning, fault tolerance, and prioritization.

Example basic configuration:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1
    maxPerZone: 5
    maxUnavailablePerZone: 1
    zones:
    - ru-central1-a
    - ru-central1-b
```

Modifying the cloud provider configuration in a cluster

The configuration of the cloud provider used in a cloud or hybrid cluster is stored in the

`<PROVIDER_NAME>ClusterConfiguration` structure, where `<PROVIDER_NAME>` is the name/code of the provider. For example, for the OpenStack provider, the structure is called OpenStackClusterConfiguration.

Regardless of the cloud provider used, its settings can be modified using the following command:

```
d8 system edit provider-cluster-configuration
```

For moving the nodes to a new operating system image, see “How do I replace the OS image in a cloud cluster?”.

NodeGroup autoscaling

In Deckhouse Platform (DP), node group autoscaling is performed based on resource demands (CPU and memory) by the `Cluster Autoscaler` component, which is part of the `node-manager` module.

Autoscaling is triggered only when there are Pending pods that cannot be scheduled on existing nodes due to insufficient resources (e.g., CPU or memory). In this case, `Cluster Autoscaler` attempts to add nodes based on the NodeGroup configuration.

Key scaling parameters are defined in the `cloudInstances` section of the NodeGroup resource:

- `minPerZone`: The minimum number of virtual machines per zone. This number is always maintained, even with no workload.
- `maxPerZone`: The maximum number of nodes that can be created per zone. This defines the upper scaling limit.
- `maxUnavailablePerZone`: Limits the number of unavailable nodes during updates, deletions, or creation.
- `standby`: Optional parameter that allows pre-provisioning standby nodes.
- `priority`: Integer priority of the group. When scaling, `Cluster Autoscaler` prefers groups with a higher `priority` value. This allows you to define scaling order among multiple NodeGroups. You can control the rate at which nodes are switched to lower-priority groups on demand (for more details, see the FAQ).

Example configuration of a NodeGroup with autoscaling:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1       # Minimum number of nodes per zone.
    maxPerZone: 5       # Maximum number of nodes per zone.
    maxUnavailablePerZone: 1  # Number of nodes that can be updated/removed
simultaneously.
    zones:
      - nova
      - supernova
      - hypernova
```

Example autoscaling scenario

Assume the following NodeGroup configuration:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: m4.large
    minPerZone: 1
    maxPerZone: 5
    zones:
      - nova
      - supernova
      - hypernova
```

There is also a Deployment with the following configuration:

```
kind: Deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: app
        resources:
          requests:
            cpu: 1500m
            memory: 5Gi
```

In this case, launching all replicas will require 3 nodes — one in each zone.

Now let's increase the number of replicas to 5. As a result, two Pods will enter the `Pending` state.

`Cluster Autoscaler`:

- Detects the situation.
- Calculates the amount of lacking resources.
- Decides to create two more nodes.
- Hands off the task to the Machine Controller Manager.
- Two new VMs will be created in the cloud and automatically join the cluster.
- The Pods will be scheduled onto the new nodes.

Allocating nodes for specific workloads

⚠ You cannot use the `deckhouse.io` domain in `labels` and `taints` keys of the NodeGroup. It is reserved for DP components. Use the `dedicated` or `dedicated.client.com` keys instead.

There are two ways to solve this problem:

1. You can set labels to NodeGroup `spec.nodeTemplate.labels`, to use them in the `Pod`'s `spec.nodeSelector` or `spec.affinity.nodeAffinity` parameters. In this case, you select nodes that the scheduler will use for running the target application.
2. You cat set taints to NodeGroup's `spec.nodeTemplate.taints` and then remove them via the `Pod`'s `spec.tolerations` parameter. In this case, you disallow running applications on these nodes unless those applications are explicitly allowed.

i DP tolerates the `dedicated` key by default, so we recommend using the `dedicated` key with any value for taints on your dedicated nodes.

To use custom keys for taints (e.g., `dedicated.client.com`), you must add the key's value to the `modules.placement.customTolerationKeys` parameter. This way, Deckhouse can deploy system components (e.g., `cni-flannel`) to these dedicated nodes.

Accelerating node provisioning in the cloud during horizontal application scaling

To speed up the launch of new application replicas during automatic horizontal scaling, it is recommended to maintain a certain number of pre-provisioned (standby) nodes in the cluster. This allows new application Pods to be scheduled quickly without waiting for node creation and initialization. Keep in mind that having standby nodes increases infrastructure costs.

The target NodeGroup configuration should be as follows:

1. Specify the absolute number of pre-provisioned nodes (or a percentage of the maximum number of nodes in the group) using the `cloudInstances.standby` parameter.
2. If there are additional service components on nodes that are not handled by Deckhouse (e.g., the `filebeat` DaemonSet), you can specify the percentage of node resources they can consume via the `standbyHolder.overprovisioningRate` parameter.
3. This feature requires that at least one group node is already running in the cluster. In other words, there must be either a single replica of the application, or the `cloudInstances.minPerZone` parameter must be set to `1`.

Example:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

Example NodeGroup definition

Cloud nodes

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
```

Configuring CloudEphemeral nodes using NodeGroupConfiguration

Additional settings for cloud nodes can be defined using NodeGroupConfiguration objects. These allow you to:

- Modify OS parameters (e.g., `sysctl`).
- Add root certificates.
- Configure trust for private image registries, etc.

NodeGroupConfiguration is applied to new nodes at creation time, including CloudEphemeral nodes.

> **i**
> NodeGroupConfiguration applies only to nodes with a specified operating system image (`bundle`).
> You can set the `bundle` value to a specific name (e.g., `ubuntu-lts`, `centos-7`, `rocky-linux`) or use `*` to apply the configuration to all OS images.

Example NodeGroupConfiguration definitions

Setting a sysctl parameter

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "*"
  content: |
    sysctl -w vm.max_map_count=262144
```

Adding a root certificate to the host

⚠

- This example is for Ubuntu OS.
  The method for adding certificates to the trust store may differ depending on the OS.
  When adapting the script to another OS, adjust the `bundles` and `content` fields accordingly.
- To use the certificate in `containerd`, you must restart the `containerd` service after adding the certificate.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )

    # bb-event           - Creating subscription for event function. More
information: http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to
the store
      update-ca-certificates
    }

    # bb-tmp-file - Creating temp file function. More information:
http://www.bashbooster.net/#tmp
    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    # bb-sync-file                                - File synchronization function.
More information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"    - Destination file
    ##  ${CERT_TMP_FILE}                          - Source file
    ##  ca-file-updated                           - Name of event that will be called
if the file changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated
```

Adding a certificate to the OS and containerd

> ⚠ This example is for Ubuntu OS.
> The method for adding certificates to the trust store may differ depending on the OS.
> When adapting the script to another OS, update the `bundles` and `content` parameters accordingly.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca-containerd.sh
```

```
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )
    CONFIG_CONTENT=$(cat <<EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
        ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
    )

    mkdir -p /etc/containerd/conf.d

    # bb-tmp-file - Create temp file function. More information:
http://www.bashbooster.net/#tmp

    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    CONFIG_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CONFIG_CONTENT}" > "${CONFIG_TMP_FILE}"

    # bb-event           - Creating subscription for event function. More
information: http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to
the store
      update-ca-certificates  # Restarting the containerd service is not required as
this is done automatically in the script 032_configure_containerd.sh
    }

    # bb-sync-file                                - File synchronization function.
More information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"    - Destination file
    ##  ${CERT_TMP_FILE}                          - Source file
    ##  ca-file-updated                           - Name of event that will be called
if the file changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated

    bb-sync-file \
      "/etc/containerd/conf.d/${REGISTRY_URL}.toml" \
      ${CONFIG_TMP_FILE}
```

Kernel update on nodes

Debian-based distros

Create a NodeGroupConfiguration resource by specifying the target kernel version in the `desired_version` variable of the shell script (the resource's `spec.content` parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

CentOS-based distros

Create a NodeGroupConfiguration resource by specifying the target kernel version in the `desired_version` variable of the shell script (the resource's `spec.content` parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="3.10.0-1160.42.2.el7.x86_64"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-dnf-install "kernel-${desired_version}"
```

Adding CloudPermanent nodes to a cloud cluster

To add `CloudPermanent` nodes to a DP cloud cluster:

1. Make sure that the corresponding cloud provider module is enabled, for example, `cloud-provider-yandex`, `cloud-provider-openstack`, or `cloud-provider-aws`.

   You can check the status of a specific module using the following command:

   ```
   d8 k -n d8-system get module <MODULE_NAME>
   ```

   For example:

   ```
   d8 k -n d8-system get module cloud-provider-yandex
   ```

2. Add a `CloudPermanent` node group to the `nodeGroups` section of the `(Provider)ClusterConfiguration` object. `CloudPermanent` nodes are managed by Terraform built into DP.

   To edit the configuration of a running cluster, use the following command:

```
d8 system edit provider-cluster-configuration
```

Example:

```
nodeGroups:
- name: cloud-permanent
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: ubuntu-24-04-cloud-amd64
    rootDiskSize: 20
    mainNetwork: default
  volumeTypeMap:
    nova: ceph-ssd
```

3. Specify the instance template parameters. The fields inside `instanceClass` depend on the cloud provider. The following fields are used in the OpenStack example:
   - `flavorName` — instance type and its resources, such as CPU and RAM
   - `imageName` — operating system image
   - `rootDiskSize` — root disk size in GB
   - `mainNetwork` — network name
   - optionally: etcd disk, zones, volume types, and other parameters

   Field names and structure may differ for other cloud providers. Refer to the `(Provider)ClusterConfiguration` specification or the documentation for the corresponding cloud provider, for example, YandexClusterConfiguration.
4. Start the DP installer container on your local computer or an administrative host. Use an image with the same edition and version as the cluster:

```
docker run --pull=always --rm -it \
  -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/<EDITION>/install:<VERSION> \
  bash
```

5. In the installer container, check the Terraform state:

```
dhctl terraform check \
  --ssh-host <MASTER_NODE_IP> \
  --ssh-user <USER_NAME> \
  --ssh-agent-private-keys /tmp/.ssh/<KEY>
```

6. Apply the configuration. For the `--ssh-user` parameter, specify a user that can connect to the master node over SSH and execute commands using `sudo`:

```
dhctl converge \
  --ssh-host <MASTER_NODE_IP> \
  --ssh-user <USER_NAME> \
  --ssh-agent-private-keys /tmp/.ssh/<KEY>
```

   This command will:
   - run Terraform
   - create the required virtual machines
   - bootstrap and configure the new nodes
   - register the nodes in the cluster
7. The new nodes will automatically appear in the cluster. You can check that the nodes have been added using the following command:

```
d8 k get nodes
```

   The new nodes are also available in the Deckhouse web interface.

Deckhouse Platform can run on top of Managed Kubernetes services, such as GKE and EKS. In this case, the `node-manager` module manages node configuration and automates node-related operations, but some features may be limited by the API of the corresponding cloud provider.

Adding a CloudStatic node to a cluster

Adding a static node can be done manually or using the Cluster API Provider Static (CAPS).

Manually

Follow the steps below to add a new static node (e.g., VM or bare metal server) to the cluster:

1. For CloudStatic nodes in the following cloud providers, refer to the steps outlined in the documentation:
   - For AWS
   - For GCP
   - For YC
2. Use the existing NodeGroup resource or create a new one. The `nodeType` parameter for static nodes in the NodeGroup must be set to `Static` or `CloudStatic`.
3. Get the Base64-encoded script code to add and configure the node.

   Here is how you can get Base64-encoded script code to add a node to the `worker` NodeGroup:

   ```
   NODE_GROUP=worker
   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} -o json | jq '.data."bootstrap.sh"' -r
   ```
4. Pre-configure the new node according to the specifics of your environment. For example:
   - Add all the necessary mount points to the `/etc/fstab` file (NFS, Ceph, etc.).
   - Install the necessary packages.
   - Configure network connectivity between the new node and the other nodes of the cluster.
5. Connect to the new node over SSH and run the following command, inserting the Base64 string you got in step 3:

   ```
   echo <Base64-CODE> | base64 -d | bash
   ```

Using the Cluster API Provider Static

A brief example of adding a static node to a cluster using Cluster API Provider Static (CAPS):

1. Prepare the necessary resources.

   - Allocate a server (or a virtual machine), configure networking connectivity, etc. If required, install specific OS packages and add the mount points on the node.
   - Create a user (`caps` in the example below) capable of executing `sudo` by running the following command **on the server**:

     ```
     useradd -m -s /bin/bash caps
     usermod -aG sudo caps
     ```
   - Allow the user to run `sudo` commands without entering a password. For this, add the following line to the `sudo` configuration **on the server** (you can either edit the `/etc/sudoers` file or run the `sudo visudo` command, or use some other method):

     ```
     caps ALL=(ALL) NOPASSWD: ALL
     ```
   - Generate a pair of SSH keys with an empty passphrase **on the server**:

     ```
     ssh-keygen -t ed25519 -f caps-id -C "" -N ""
     ```

     The public and private keys of the `caps` user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.
   - Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory **on the server**:

```
mkdir -p /home/caps/.ssh
cat caps-id.pub >> /home/caps/.ssh/authorized_keys
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps/
```

2. Create an SSHCredentials resource in the cluster:

   Run the following command in the user key directory **on the server** to encode the private key to Base64:

   ```
   base64 -w0 caps-id
   ```

   On any computer with `d8 k` configured to manage the cluster, create an environment variable with the value of the Base64-encoded private key you generated in the previous step:

   ```
   CAPS_PRIVATE_KEY_BASE64=<BASE64-ENCODED PRIVATE KEY>
   ```

   To create an SSHCredentials resource in the cluster (note that from this point on, you have to use `d8 k` configured to manage the cluster), run the following command:

   ```
   d8 k create -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: SSHCredentials
   metadata:
     name: credentials
   spec:
     user: caps
     privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
   EOF
   ```

3. Create a StaticInstance resource in the cluster and specify the IP address of the static node server:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: StaticInstance
   metadata:
     name: static-worker-1
     labels:
       role: worker
   spec:
     # Specify the IP address of the static node server.
     address: "<SERVER-IP>"
     credentialsRef:
       kind: SSHCredentials
       name: credentials
   EOF
   ```

4. Create a NodeGroup resource in the cluster. The value of `count` defines a number of `staticInstances` which fall under the `labelSelector` that will be added to the cluster. In this example it's `1`:

   > **i** The `labelSelector` field in the NodeGroup resource is immutable. To update the `labelSelector`, you need to create a new NodeGroup and move the static nodes into it by changing their labels.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

Using Cluster API Provider Static for multiple node groups

This example shows how you can use filters in the StaticInstance `label selector` to group static nodes and use them in different NodeGroups. Here, two node groups (`front` and `worker`) are used for different tasks. Each group includes nodes with different characteristics — the `front` group has two servers and the `worker` group has one.

1. Prepare the required resources (3 servers or virtual machines) and create the SSHCredentials resource the same way as on step 1 and step 2 of the example.
2. Create two NodeGroup resources in the cluster (from this point on, use `d8 k` configured to manage the cluster):

   > **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new NodeGroup and move the static nodes into it by changing their labels.

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: front
   spec:
     nodeType: Static
     staticInstances:
       count: 2
       labelSelector:
         matchLabels:
           role: front
   ---
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     nodeType: Static
     staticInstances:
       count: 1
       labelSelector:
         matchLabels:
           role: worker
   EOF
   ```

3. Create StaticInstance resources in the cluster and specify the valid IP addresses of the servers:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

Adding master nodes in a cloud cluster

To add master nodes in a cloud cluster:

1. Make sure the `control-plane-manager` module is enabled.
2. Open the ClusterConfiguration file (e.g., OpenStackClusterConfiguration).
3. Add or update the `masterNodeGroup` section:

   ```
   masterNodeGroup:
     replicas: 3
     instanceClass:
       flavorName: m1.medium
       imageName: ubuntu-24-04-cloud-amd64
       rootDiskSize: 20
       mainNetwork: default
   ```

4. Apply the changes using `dhctl converge`:

   ```
   dhctl converge \
     --ssh-host <master node IP> \
     --ssh-user <username> \
     --ssh-agent-private-keys /tmp/.ssh/<key>
   ```

Using NodeGroup with priority

The `priority` parameter of the NodeGroup custom resource allows you to define the order in which nodes are provisioned in the cluster.
For example, you can configure the cluster to first provision *spot-node* instances and fall back to regular nodes if spot capacity runs out.
Or you can instruct the system to prioritize larger nodes when available, and use smaller ones when resources are limited.

Example of creating two NodeGroups using spot-node instances:

```
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral
```

In the example above, the `Cluster Autoscaler` will first attempt to provision a spot-node. If it fails to add the node to the cluster within 15 minutes, the `worker-spot` NodeGroup will be paused for 20 minutes, and the `Cluster Autoscaler` will start provisioning nodes from the `worker` NodeGroup.

If a new node is needed again after 30 minutes, `Cluster Autoscaler` will again attempt `worker-spot` first, then fall back to `worker`.

Once the `worker-spot` NodeGroup reaches its maximum (5 nodes in the example), all further nodes will be provisioned from the `worker` NodeGroup.

The node templates (labels/taints) for `worker` and `worker-spot` NodeGroups should be identical or at least compatible with the workload that triggers scaling.

NodeGroup states and their interpretation

Ready — the NodeGroup contains the minimum required number of scheduled nodes with the status `Ready` for all zones.

Example 1. A group of nodes in the `Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
  - status: "True"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Example 2. A group of nodes in the `Not Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
  - status: "False"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Updating: A node group contains at least one node in which there is an annotation with the prefix `update.node.deckhouse.io` (for example, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval: A node group contains at least one node that has an annotation `update.node.deckhouse.io/disruption-required` and there is no annotation `update.node.deckhouse.io/disruption-approved`.

Scaling: Calculated only for node groups with the type `CloudEphemeral`. The state `True` can be in two cases:

1. When the number of nodes is less than the *target number of nodes* in the group, i.e. when it is

necessary to increase the number of nodes in the group.
2. When a node is marked for deletion or the number of nodes is greater than the *target number of nodes*, i.e. when it is necessary to reduce the number of nodes in the group.

The *target number of nodes* is the sum of all replicas in the node group.

Example. The desired number of nodes is 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
...
  desired: 2
...
```

Error: Contains the last error that occurred when creating a node in a node group.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `cloud-provider-gcp` documentation
- Module `cloud-provider-openstack` documentation
- Module `cloud-provider-yandex` documentation
- Module `control-plane-manager` documentation
- Module `node-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Adding and managing bare-metal nodes

For more details on sizing resources for a bare-metal cluster, see the “Hardware requirements for bare-metal cluster” guide.

Adding nodes to a bare-metal cluster

Manual method

1. Enable the `node-manager` module.
2. Create a NodeGroup object with the type `Static`:

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     nodeType: Static
   ```

   In this resource, specify the `Static` node type. For all NodeGroup objects in the cluster, Deckhouse automatically generates a `bootstrap.sh` script used to add nodes to the group. When adding nodes manually, you need to copy this script to the server and run it.

   You can obtain the script from the Deckhouse web interface under the “ Node Groups → Scripts”

tab or via the following `d8 k` command:

```
d8 k -n d8-cloud-instance-manager get secrets manual-bootstrap-for-worker -ojsonpath="
{.data.bootstrap\.sh}"
```

The script needs to be decoded from Base64 and then executed as `root`.

3. Once the script finishes, the server will be added to the cluster as a node in the specified group.

Automatic method

⚠ If you have previously increased the number of master nodes in the cluster in the NodeGroup `master` (parameter `spec.staticInstances.count`), before adding nodes using automatic method, make sure that they will not be “captured”.

DP supports automatic addition of physical (bare-metal) servers to the cluster without the need to manually run an installation script on each node. To enable this:

1. Prepare the server (OS, networking):
   - Install a supported operating system.
   - Configure networking and ensure the server is accessible via SSH.
   - Create a system user (e.g., `ubuntu`) for SSH access.
   - Ensure the user can execute commands using `sudo`.
2. Create an SSHCredentials object to define access to the server. DP uses this object to connect to the server over SSH. It specifies:
   - A private SSH key encoded in Base64 format.
   - The OS user.
   - The SSH port.
   - (Optional) A `sudo` password, if required.

   Example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: SSHCredentials
   metadata:
     name: static-nodes
   spec:
     privateSSHKey:
   LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KTUlJRXZBSUJBREFOQmdrcWhraUc
   5dzBCQVFFRkFBT0NBZzhBTUlJQkNnS0NBUUVB
     sshPort: 22
     sudoPassword: password
     user: ubuntu
   ```

   i **Important**. The `privateSSHKey` field must contain a private SSH key encoded in Base64 format. The private key must match the corresponding public key added to the `~/.ssh/authorized_keys` file on the server.

   For details on SSHCredentials sensitive data protection, see the `node-manager` module FAQ.
3. Create a StaticInstance` object for each server:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-0
  labels:
    static-node: auto
spec:
  address: 192.168.1.10
  credentialsRef:
    apiVersion: deckhouse.io/v1alpha1
    kind: SSHCredentials
    name: static-nodes
```

A separate StaticInstance resource must be created for each server, but the same SSHCredentials can be reused to access multiple servers.

Possible StaticInstance states:

- `Pending`: The server has not yet been configured; the corresponding node is not present in the cluster.
- `Bootstrapping`: The server is being configured and the node is being added to the cluster.
- `Running`: The server is successfully configured and the node has joined the cluster.
- `Cleaning`: The server is being cleaned up and the node is being removed from the cluster.

These states reflect the current stage of node management. CAPS automatically transitions a StaticInstance between these states depending on whether a node needs to be added or removed from a group.

4. Create a NodeGroup resource describing how DP should use these servers:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 3
    labelSelector:
      matchLabels:
        static-node: auto
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
```

This section defines parameters for using StaticInstance resources:

- `count` specifies how many nodes will be added to the group.
- `labelSelector` defines the rules for selecting nodes.

When using the Cluster API Provider Static (CAPS), it is important to correctly set the `nodeType` to `Static` and provide the `staticInstances` section in the NodeGroup resource:

- If the `labelSelector` is not specified, CAPS will use any available StaticInstance resources in the cluster.
- The same StaticInstance can be used in multiple NodeGroups if it matches the filters.
- CAPS automatically maintains the number of nodes in the group according to the `count` parameter.
- When a node is removed, CAPS performs cleanup and disconnection, and the corresponding StaticInstance transitions to the `Pending` status, allowing it to be reused.

After the node group is created, a script for adding servers to the group will become available. DP will wait for the required number of StaticInstance objects that match the specified labels. As soon as such an object appears, DP will use the provided IP address and SSH connection parameters to run the `bootstrap.sh` script and add the server to the group.

Creating nodes through BMC

If servers have baseboard management controllers (BMCs) with Redfish or IPMI support, Deckhouse Platform can install an operating system on them and join them to the cluster by itself. The `infra-provider-baremetal` module provides this capability. With it, `node-manager` creates nodes on such servers in node groups with the `CloudEphemeral` type and scales them automatically.

The setup procedure is described on the Usage page of the `infra-provider-baremetal` module.

Modifying a static cluster configuration

The static cluster settings are stored in the StaticClusterConfiguration structure.

To modify the static cluster parameters, run the following command:

```
d8 system edit static-cluster-configuration
```

Moving a static node between NodeGroups

⚠ During the migration of static nodes between NodeGroups, the node is cleaned up and bootstrapped again, and the `Node` object is recreated.

1. Create a new NodeGroup resource, for example named `front`, which will manage the static node labeled `role: front`:

   ```
   d8 k create -f - <<EOF
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: front
   spec:
     nodeType: Static
     staticInstances:
       count: 1
       labelSelector:
         matchLabels:
           role: front
   ```

2. Change the `role` label of the existing StaticInstance from `worker` to `front`.
   This will allow the new NodeGroup named `front` to manage this node:

   ```
   d8 k label staticinstance static-worker-1 role=front --overwrite
   ```

3. Update the `worker` NodeGroup resource by decreasing the `count` parameter from `1` to `0`:

   ```
   d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --
   type=merge
   ```

Manual cleanup of a static node

To remove a node from the cluster and clean the server, run the `/var/lib/bashible/cleanup_static_node.sh` script, which is already present on every static node.

Example of disconnecting a node from the cluster and cleaning the server:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-
doing
```

ℹ This instruction applies both to nodes manually configured using the bootstrap script and to nodes configured via CAPS.

NodeGroup example

Example NodeGroup definition for static nodes

For virtual machines on hypervisors or physical servers, use static nodes by setting `nodeType: Static` in the NodeGroup.

Example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Nodes are added to such a group manually using preconfigured scripts or automatically via CAPS.

Changing the CRI for a NodeGroup

CRI (Container Runtime Interface) is a standard interface between the kubelet and the container runtime.

> ⚠ CRI can only be switched between `Containerd` and `NotManaged` via the `cri.type` parameter.

To change the CRI for a NodeGroup, set the `cri.type` parameter to either `Containerd` or `NotManaged`.

Example YAML manifest:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

You can also perform this operation using a patch:

- To set `Containerd`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":
  {"type":"Containerd"}}}'
  ```

- To set `NotManaged`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":
  {"type":"NotManaged"}}}'
  ```

> ⚠ When changing the `cri.type` for a NodeGroup created using `dhctl`, you must also update this value in `dhctl config edit provider-cluster-configuration` and in the NodeGroup object settings.

After changing the CRI for a NodeGroup, the `node-manager` module will sequentially reboot the nodes, applying the new CRI.
Node updates involve disruption. Depending on the `disruption` settings for the NodeGroup, the `node-manager` module will either automatically update the nodes or require manual approval.

Changing the NodeGroup of a static node

If a node is managed by CAPS, you can’t change its associated NodeGroup.
The only option is to delete the StaticInstance and create a new one.

To move an existing manually added static node from one NodeGroup to another, you need to update the group label on the node:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

It will take some time for the changes to take effect.

Changing the IP address in a StaticInstance

You cannot change the IP address of a StaticInstance resource.
If an incorrect address is specified in a StaticInstance, you need to delete the StaticInstance and create a new one.

Deleting a StaticInstance

A StaticInstance in the `Pending` state can be safely deleted without any issues.

To delete a StaticInstance that is in any state other than `Pending` (`Running`, `Cleaning`, `Bootstrapping`), follow these steps:

1. Add the label `"node.deckhouse.io/allow-bootstrap": "false"` to the StaticInstance.

   Example command for adding a label:

   ```
   d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
   ```

2. Wait until the StaticInstance transitions to the `Pending` state.

   To check the status of StaticInstance, use the command:

   ```
   d8 k get staticinstances
   ```

3. Delete the StaticInstance.

   Example command for deleting StaticInstance:

   ```
   d8 k delete staticinstance d8cluster-worker
   ```

4. Decrease the `NodeGroup.spec.staticInstances.count` parameter by 1.

Additional resources

- Global custom resources
- Module `infra-provider-baremetal` documentation
- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom node configuration

To automate actions on group nodes, use the NodeGroupConfiguration resource. It allows you to run bash scripts on the nodes using the Bash Booster command set, as well as apply the Go Template templating engine. This is useful for automating operations such as:

- Installing and configuring additional OS packages.

  Examples:

  - Installing the kubectl plugin

- Updating the OS kernel to a specific version.

Examples:

  - Debian kernel update
  - CentOS kernel update

- Modifying OS parameters.

  Examples:

  - Tuning a sysctl parameter
  - Adding a root certificate

- Collecting information on the node and performing similar tasks.

The NodeGroupConfiguration resource allows you to define execution priority for scripts, limit execution to specific node groups or OS types.

The script code is specified in the `content` field of the resource. When a script is created on a node, the content passes through the Go Template templating engine, which adds a layer of logic to script generation. A dynamic context with a set of variables is available in the template.

Available template variables include:

- `.cloudProvider` (for node groups with `CloudEphemeral` or `CloudPermanent` nodeType) — an array of cloud provider data.

  Example data...

  ```
  cloudProvider:
    instanceClassKind: OpenStackInstanceClass
    machineClassKind: OpenStackMachineClass
    openstack:
      connection:
        authURL: https://cloud.provider.com/v3/
        domainName: Default
        password: p@ssw0rd
        region: region2
        tenantName: mytenantname
        username: mytenantusername
      externalNetworkNames:
      - public
      instances:
        imageName: ubuntu-24-04-cloud-amd64
        mainNetwork: kube
        securityGroups:
        - kube
        sshKeyPairName: kube
      internalNetworkNames:
      - kube
      podNetworkMode: DirectRoutingWithPortSecurityEnabled
    region: region2
    type: openstack
    zones:
    - nova
  ```

- `.cri` — the container runtime interface in use (since Deckhouse version 1.49, only `Containerd` is used).
- `.kubernetesVersion` — the version of Kubernetes in use.
- `.nodeUsers` — an array of user data added to the node using the NodeUser resource.

  Example data...

```
nodeUsers:
- name: user1
  spec:
    isSudoer: true
    nodeGroups:
    - '*'
    passwordHash: PASSWORD_HASH
    sshPublicKey: SSH_PUBLIC_KEY
    uid: 1050
```

- `.nodeGroup` — an array of NodeGroup data.

  Example data...

  ```
  nodeGroup:
    cri:
      type: Containerd
    disruptions:
      approvalMode: Automatic
    kubelet:
      containerLogMaxFiles: 4
      containerLogMaxSize: 50Mi
      resourceReservation:
        mode: "Off"
    kubernetesVersion: "1.31"
    manualRolloutID: ""
    name: master
    nodeTemplate:
      labels:
        node-role.kubernetes.io/control-plane: ""
        node-role.kubernetes.io/master: ""
      taints:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
    nodeType: CloudPermanent
    updateEpoch: "1699879470"
  ```

Example of using variables in the template engine:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

Example of using Bash Booster commands:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Monitoring script execution

Applying NodeGroupConfiguration triggers an update of the **`bashible` configuration of the node group** in the specified groups. Deckhouse Platform automatically detects the change in this configuration and then sets the `UPTODATE` field (in the output of the `d8 k get nodegroup` command) to `0`. The `UPTODATE` field shows how many nodes have already been brought to the **target `bashible` configuration of the node group**; a value of `0` means that no node has been brought to the target `bashible` configuration yet (i.e., the update has not been applied to any node).

Deckhouse Platform controls the start of updates on nodes via the `update.node.deckhouse.io/approved` annotation on the `Node` object. By default, the update runs

simultaneously on one node from each group. Update parallelism is defined by the `maxConcurrent` parameter in the node group configuration. When Deckhouse Platform selects a node to update (taking the queue and `maxConcurrent` into account), it sets the `update.node.deckhouse.io/approved` annotation, after which the `bashible` service on that node begins applying the **target `bashible` configuration of the node group**.

You can view the node annotations set by Deckhouse Platform to start the update using the following command:

```
d8 k get nodes -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") |
.metadata.name' -r
```

You can monitor script execution on the node in the `bashible` service logs using:

```
journalctl -fu bashible.service
```

The scripts are located in the `/var/lib/bashible/bundle_steps/` directory on the node.

Script re-execution mechanism

The service decides whether to re-run the scripts by comparing a unified checksum of all files, located at `/var/lib/bashible/configuration checksum`, with the checksum stored in the `configuration-checksums` secret in the `d8-cloud-instance-manager` namespace in the Kubernetes cluster.

You can check the checksum with the following command:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o yaml
```

The checksum comparison is performed by the service every minute.

The checksum in the cluster is updated every 4 hours, thereby re-triggering the execution of scripts on all nodes.

To manually trigger `bashible` execution on a node, you can delete the checksum file using the following command:

```
rm /var/lib/bashible/configuration_checksum
```

Script writing specifics

When writing scripts, it’s important to consider the following features of their usage in Deckhouse:

1. Scripts in Deckhouse Platform are executed every 4 hours or based on external triggers. Therefore, it’s important to write scripts in a way that they first check whether changes are necessary, to avoid repeated or unnecessary actions on each execution.
2. There are predefined scripts that perform various actions, including service installation and configuration. It’s important to consider this when assigning priority to custom scripts. For example, if a custom script restarts a service, it must run after the script that installs that service. Otherwise, the custom script won’t be able to run during the initial provisioning of the node (since the service won’t be installed yet).

Useful specifics of certain scripts:

- `032_configure_containerd.sh`: Merges the containerd configuration files and **restarts** the service. The location of the configuration files depends on the CRI used on the cluster nodes:
  - `/etc/containerd/conf.d/*.toml`: For containerd v1.
  - `/etc/containerd/conf2.d/*.toml`: For containerd v2.

  Note that directories containing configuration files are not created automatically. Files must be added to these directories in scripts with a priority of less than `32`.

How to use containerd with Nvidia GPU support

On GPU nodes, containerd is configured by the `gpu` module: it adds the `nvidia` runtime and the required system settings to the nodes of the node groups with the `spec.gpu` section. Do not configure the `nvidia` runtime with your own NodeGroupConfiguration resources: such a configuration conflicts with the one of the `gpu` module.

How to install the NVIDIA driver and NVIDIA Container Toolkit on the nodes and how to verify a GPU node is described in the `gpu` module FAQ.

How to deploy a custom containerd configuration file

✖ Adding custom settings will trigger a restart of the `containerd` service.

Bashible on nodes merges main Deckhouse containerd config with configs from:

- `/etc/containerd/conf.d/*.toml` — if containerd v1 is used as the CRI on the cluster nodes.
- `/etc/containerd/conf2.d/*.toml` — if containerd v2 is used as the CRI on the cluster nodes.

⚠ You can override the parameter values defined in the `/etc/containerd/deckhouse.toml` file. However, you are responsible for ensuring the correct operation of such changes. It is recommended **not to modify the configuration** on control plane (master) nodes (NodeGroup `master`).

- For containerd v1
- For containerd v2

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
      [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-
shm.options]
        BinaryName = ""
        SystemdCgroup = true
        ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

Adding a configuration for an additional registry

Containerd supports two methods for registry configuration: the **deprecated** method and the **actual** method.

To check for the presence of the **deprecated** configuration method, run the following commands on the cluster nodes:

```
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.configs'
```

Example output:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
[plugins."io.containerd.grpc.v1.cri".registry.configs]
  [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]
```

To check for the presence of the **actual** configuration method, run the following command on the cluster nodes:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'

# Example output:
# config_path = "/etc/containerd/registry.d"
```

⚠ For more configuration examples for additional registry (for example, deploying a custom containerd configuration file), see the “FAQ” section of the `node-manager` module documentation.

Old Method

⚠ This containerd configuration format is deprecated.

i Used in containerd v1 when Deckhouse is not managed by the Registry module.

The configuration is described in the main containerd configuration file `/etc/containerd/config.toml`.

Adding custom configuration is carried out through the `toml merge` mechanism. Configuration files from the `/etc/containerd/conf.d` directory are merged with the main file `/etc/containerd/config.toml`. The merge takes place during the execution of the `032_configure_containerd.sh` script, so the corresponding files must be added in advance.

Example configuration file for the `/etc/containerd/conf.d/` directory:

```
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
          endpoint = ["https://${REGISTRY_URL}"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
          auth = "${BASE_64_AUTH}"
          username = "${USERNAME}"
          password = "${PASSWORD}"
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
          ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
          insecure_skip_verify = true
```

✖ Adding custom settings through the `toml merge` mechanism causes the containerd service to restart.

Adding authorization to an additional registry (deprecated method)

Example of adding authorization to an additional registry when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

Configuring a certificate for an additional registry (deprecated method)

Example of configuring a certificate for an additional registry when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"


    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
            ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
```

i In addition to containerd, the certificate can be added to the OS.

Adding TLS skip verify (deprecated method)

Example of adding TLS skip verify when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
            insecure_skip_verify = true
    EOF
```

After applying the configuration file, verify access to the registry from the nodes using the command:

```
# Via the CRI interface
crictl pull private.registry.example/image/repo:tag
```

Configuring a mirror for access to public registries (deprecated method)

Example of configuring a mirror for public image registries when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
              endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
              endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]
    EOF
```

New Method

> **i**
>
> Used in containerd v2.
>
> Used in containerd v1 when managed through the `registry` module (for example, in `Direct` mode).

The configuration is defined in the `/etc/containerd/registry.d` directory.
Configuration is specified by creating subdirectories named after the registry address:

```
/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.io
    ├── ca.crt
    └── hosts.toml
```

Example contents of the `hosts.toml` file:

```
[host]
  # Mirror 1.
  [host."https://${REGISTRY_URL_1}"]
    capabilities = ["pull", "resolve"]
    ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

    [host."https://${REGISTRY_URL_1}".auth]
      username = "${USERNAME}"
      password = "${PASSWORD}"

  # Mirror 2.
  [host."http://${REGISTRY_URL_2}"]
    capabilities = ["pull", "resolve"]
    skip_verify = true
```

i Configuration changes do not cause the containerd service to restart.

Adding authorization to an additional registry (actual method)

Example of adding authorization to an additional registry when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        [host."https://${REGISTRY_URL}".auth]
          username = "username"
          password = "password"
    EOF
```

Configuring a certificate for an additional registry (actual method)

Example of configuring a certificate for an additional registry when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        ca = ["/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt"]
    EOF
```

i In addition to containerd, the certificate can be added to the OS.

Adding TLS skip verify (actual method)

Example of adding TLS skip verify when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        skip_verify = true
    EOF
```

After applying the configuration file, check access to the registry from the nodes using the following commands:

```
# Via the CRI interface.
crictl pull private.registry.example/image/repo:tag

# Via ctr with the configuration directory specified.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/image/repo:tag

# Via ctr for an HTTP registry.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/image/repo:tag
```

Configuring a mirror for access to public registries (actual method)

Example of configuring a mirror for public image registries when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
```

How to automatically put custom labels on the node

1. On the node, create the directory `/var/lib/node_labels`.
2. In this directory, create a single of multiple files containing the necessary labels. There can be any number of files, as well as the number of subdirectories containing them.
3. Add the necessary labels to the files in the `key=value` format. For example:

   ```
   example-label=test
   ```

4. Save the files.

When adding a node to the cluster, the labels specified in the files will be automatically affixed to the node.

> ⚠ Note that it is not possible to add labels used in Deckhouse Platform in this way. This method will only work with custom labels that do not overlap with those reserved for Deckhouse.

Additional resources

- Module `deckhouse` parameters

- Module `gpu` documentation
- Module `node-manager` documentation
- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Migrating container runtime to containerd v2

You can configure containerd v2 as the primary container runtime either at the cluster level or for specific node groups. This runtime option enables the use of cgroups v2, provides improved security, and allows more flexible resource management.

> To view the current container runtime type used by default in the cluster NodeGroups, use the following command:
>
> ```
> d8 system edit cluster-configuration
> ```
>
> The container runtime type is specified in the `defaultCRI` parameter.

Requirements

Migration to containerd v2 is possible under the following conditions:

- Nodes meet the requirements described in the cluster-wide parameters.
- The node kernel version is not in the range 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- There are no custom configurations on the server in `/etc/containerd/conf.d` (example of a custom configuration).

> ⚠ On Ubuntu nodes in AWS, the `erofs` kernel module required by containerd v2 is shipped in a separate `linux-modules-extra` package. Deckhouse Platform installs it automatically, but for some kernels, mostly outdated ones, the package is no longer published in the APT repository. In that case, the installation is skipped, a warning is written to the node logs, the node gets the `node.deckhouse.io/containerd-v2-unsupported` label, and the `D8NodeContainerdV2NotSupported` alert fires.
>
> The problem has been confirmed for the `5.15.0-1028-aws`, `6.8.0-1024-aws`, and `6.8.0-1029-aws` kernels. To migrate such nodes, switch the node group to a newer AMI or kernel that provides `erofs`.
>
> Before migrating, check whether there are such nodes in the cluster: check the alerts and the list of labelled nodes (see the commands below).

If any of the requirements described in the general cluster parameters are not met, Deckhouse Platform adds the label `node.deckhouse.io/containerd-v2-unsupported` to the node. If the node has custom configurations in the `/etc/containerd/conf.d/` (if CRI containerd v1 is used on the cluster nodes) or `/etc/containerd/conf2.d/` (if CRI containerd v2 is used on the cluster nodes) directories, the label `node.deckhouse.io/containerd-config=custom` is added to it.

If one of these labels is present, changing the `spec.cri.type` parameter for the node group will be unavailable. Nodes that do not meet the migration conditions can be viewed using the following commands:

```
d8 k get node -l node.deckhouse.io/containerd-v2-unsupported
d8 k get node -l node.deckhouse.io/containerd-config=custom
```

Additionally, a administrator can verify if a specific node meets the requirements using the following commands:

```
uname -r | cut -d- -f1
stat -f -c %T /sys/fs/cgroup
systemctl --version | awk 'NR==1{print $2}'
modprobe -qn erofs && echo "TRUE" || echo "FALSE"
ls -l /etc/containerd/conf.d
```

How to enable containerd v2

> **i** During migration the directory `/var/lib/containerd` will be cleared, causing all pod images to be re-pulled, and the node will reboot.
>
> Also note that this is a disruptive update. If `spec.disruptions.approvalMode: Manual` is set for a node group, approve the update on the respective nodes with the annotation: `d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=`.

You can enable containerd v2 in two ways:

1. **For the entire cluster**. Set the value `ContainerdV2` for the `defaultCRI` parameter in the `ClusterConfiguration` resource. This value will apply to all NodeGroup objects where `spec.cri.type` is not explicitly defined.

   Example:

   ```
   apiVersion: deckhouse.io/v1
   kind: ClusterConfiguration
   ...
   defaultCRI: ContainerdV2
   ```

2. **For a specific node group**. Set `ContainerdV2` in the `spec.cri.type` parameter of the NodeGroup object.

   Example:

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     cri:
       type: ContainerdV2
   ```

   When migrating to containerd v2 Deckhouse Platform will begin sequentially updating the nodes.

Additional resources

- Global custom resources
- Global custom resources
- Module `node-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Maintenance of nodes running virtual machines

Work on a node affects the virtual machines (VMs) running on it. The sections below cover how to move machines to other nodes, how to take a node out for maintenance, and what to do with machines that cannot be moved.

VM migration and node maintenance

Live migration moves a running virtual machine from one node to another without shutting it down. It’s needed in three situations:

- Load balancing, to distribute VMs evenly across nodes.
- Node maintenance or update, to free the node from VMs.
- Virtual machine firmware update, which would otherwise require a restart.

> **i** Live migration is limited in speed and in the number of concurrent moves:
>
> - A node prepares and sends the memory of only one VM at a time, and accepts only one incoming migration at a time.
> - This also sets the cluster limit: no more concurrent migrations than there are nodes allowed to run virtual machines.
> - The transfer rate of a single migration is limited to 640 MiB/s, which is about 5 Gbit/s.

Moving a selected VM to another node

The following steps show how to move a selected VM to another node.

- Using the CLI
- Using the web interface

1. Check which node the VM currently runs on:

   ```
   d8 k get vm
   ```

   Example output:

   ```
   NAME       PHASE     UPTIME   NODE           IPADDRESS     AGE
   linux-vm   Running   79m      virtlab-pt-1   10.66.10.14   79m
   ```

   The VM runs on the `virtlab-pt-1` node.

2. Create a VirtualMachineOperation resource with the `Evict` type. Deckhouse Platform (DP) selects a new node for the VM, respecting its placement requirements:

   ```
   d8 k create -f - <<EOF
   apiVersion: virtualization.deckhouse.io/v1alpha2
   kind: VirtualMachineOperation
   metadata:
     generateName: evict-linux-vm-
   spec:
     # Virtual machine name.
     virtualMachineName: linux-vm
     # Operation for the migration.
     type: Evict
   EOF
   ```

3. Right after creating the resource, follow the migration progress:

   ```
   d8 k get vm -w
   ```

   Example output:

   ```
   NAME       PHASE       UPTIME   NODE           IPADDRESS     AGE
   linux-vm   Running     79m      virtlab-pt-1   10.66.10.14   79m
   linux-vm   Migrating   79m      virtlab-pt-1   10.66.10.14   79m
   linux-vm   Migrating   79m      virtlab-pt-1   10.66.10.14   79m
   linux-vm   Running     79m      virtlab-pt-2   10.66.10.14   79m
   ```

   The VM keeps its IP address during the move; only the node in the `NODE` column changes.

4. To interrupt the migration, delete the created resource while it's in the `Pending` or `InProgress` phase.

1. Go to the **Projects** tab and select the project you need.
2. Go to **Virtualization → Virtual machines**.
3. Select the virtual machine from the list and click the ellipsis button.
4. In the menu that opens, select **Migrate**.
5. In the **Virtual machine migration** window, select **Migrate to an arbitrary node** or **Migrate to a selected node** and specify the node in the **Nodes available for migration** field.
6. If required, enable **Migrate disks** to move the VM disks along with the VM, and **Force (slow down guest CPU)** to make sure the migration completes for an actively running VM.
7. Click **Migrate**, or cancel the operation with **Cancel**.

Dedicated migration network

By default, live migration traffic goes over the main node network and competes for bandwidth with workloads. You can route it through a dedicated VLAN provided by the `sdn` module.

This requires the `sdn` module to be enabled and a SystemNetwork resource to be created and in the `Ready` state.

To route the traffic to the dedicated network, set the `.spec.settings.liveMigration.network` block in the `virtualization` ModuleConfig. Specify `type: SystemNetwork` in it and the name of the prepared network in the `systemNetwork.name` field. After that, all migrations in the cluster go over the VLAN of that network.

```
spec:
  settings:
    liveMigration:
      network:
        type: SystemNetwork
        systemNetwork:
          name: migration-net
```

To return migration traffic to the main node network, delete the `network` block. When it isn't set, the node network is used by default.

To create a system network in the web interface:

1. Go to the **System** tab, then to **Network → SDN → System networks**.
2. Click **Create**.
3. In the **Create resource** window that opens, enter the network name in the **Name** field.
4. On the **Configuration** tab, select the type (`VLAN`, `Access`, or `SRIOVVirtualFunction`) in the **Type** field, the underlay network in the **Underlay network** field, and the VLAN identifier in the **VLAN** block. Set the **IPAM** block parameters, if required.
5. Click **Apply**.
6. Review the created networks in the list, which shows the **Status**, **Type**, **Underlay network**, **VLAN ID**, and **IP pool** columns.

Beforehand, create a network class (VLAN ID ranges and parent network interfaces of nodes) and an underlay network (participating network interfaces of nodes and the **Dedicated** or **Shared** mode) in the **Network → SDN → Network classes** and **Underlay networks** sections.

Checking VMs before node maintenance

It's better to find the VMs that won't be able to migrate off a node before maintenance starts, before the node is made unschedulable:

```
d8 k get vm -o wide | grep <NODE_NAME>
```

VMs with the `False` value in the `MIGRATABLE` column have to be stopped when the node is drained. Live migration isn't possible for them, and the evacuation fails.

The column value alone isn't enough. A VM with the `True` value and the `VirtualMachineWaitingForMigrationTarget` reason won't move anywhere either, until a suitable node returns to scheduling, so before maintenance, look at the reasons for all VMs on the node:

```
d8 k get vm -o json | jq -r '.items[] | [.metadata.name, (.status.conditions[] |
select(.type=="Migratable") | .reason)] | @tsv'
```

Maintenance mode

Work on a node that runs virtual machines can disrupt them. To prevent this, switch the node to maintenance mode, and DP moves the VMs to other nodes.

- Using the CLI
- Using the web interface

To free the node from all resources, including system ones, run:

```
d8 k drain <NODE_NAME> --ignore-daemonsets --delete-emptydir-data
```

To evict only virtual machines from the node, add a label selector:

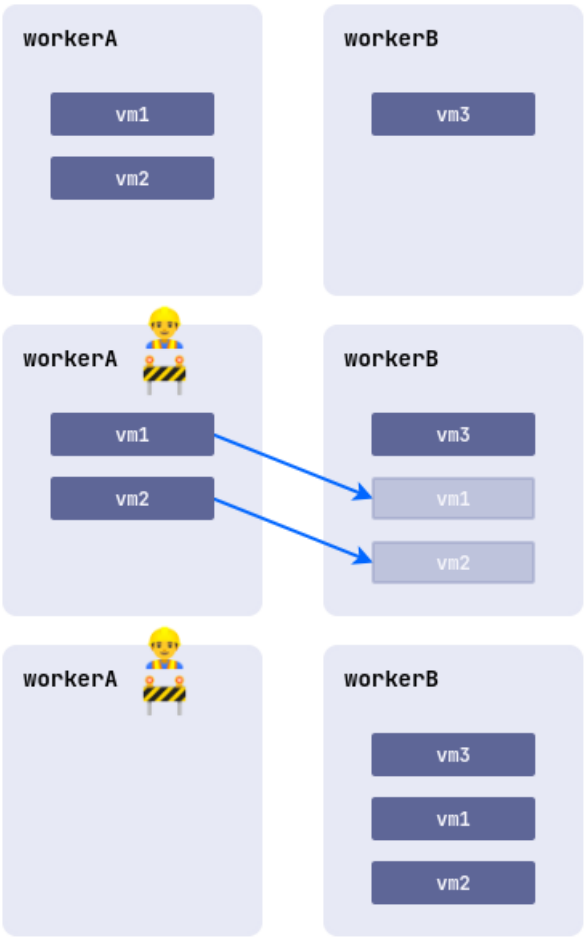
```
d8 k drain <NODE_NAME> --pod-selector vm.kubevirt.internal.virtualization.deckhouse.io/name --
delete-emptydir-data
```

Where `<NODE_NAME>` is the name of the node scheduled for maintenance.

After the command runs, the node switches to maintenance mode, and virtual machines can’t be started on it.

To return the node to service, stop the `drain` command with `Ctrl+C`, and then run:

```
d8 k uncordon <NODE_NAME>
```



1. Go to the **System** tab, then to **Nodes**.
2. Select the node from the list, click the ellipsis button, and select **Cordon + Drain** in the menu that opens.
3. To take the node out of maintenance mode, select **Uncordon** in the same menu.

Restarting virtual machines during node maintenance

A virtual machine can’t always be moved to another node by live migration. It can be pinned to the node by placement rules or use a device passed through from the node. The `Migratable` condition in

the VM status shows the reason. Such a VM keeps running and holds the node, so maintenance can’t complete until the VM is restarted.

When DP finds such a VM while switching the node to maintenance mode, it adds the `virtualization.deckhouse.io/virtualmachines-restart-required` annotation to the node. To allow the restart, add the matching annotation to the node:

```
d8 k annotate node <NODE_NAME> virtualization.deckhouse.io/virtualmachines-restart-approved=""
```

Where `<NODE_NAME>` is the name of the node being switched to maintenance mode.

Only the VMs that can’t be moved by live migration are restarted. The guest OS shuts down gracefully, after which the VM starts according to the `.spec.runPolicy` run policy. For each restart, a VirtualMachineOperation resource named `node-maintenance-restart-*` is created. A restart interrupts the applications inside the VM, so coordinate it with the project owners.

The approval doesn’t apply to VMs that can be moved by live migration, including those with no suitable node at the moment. Such VMs are moved by live migration as soon as a suitable node appears.

You can grant the approval in advance, while planning the work. Until the node is switched to maintenance mode, the annotation has no effect. DP removes both annotations once the node is free, so one approval covers one maintenance of one node.

DP reacts to VM eviction from a node. If eviction stopped on timeout (the `.spec.nodeDrainTimeoutSecond` parameter of the NodeGroup resource, 10 minutes by default), eviction isn’t retried. An approval granted after that doesn’t trigger a restart, and you have to free the node manually.

A restart frees the node but doesn’t guarantee that the VM starts on another one right away. The limitation that prevents live migration usually prevents the start on another node as well. In that case, the VM stays in the `Pending` phase, and the `Running` condition reports the reason received from the scheduler. Maintenance can continue meanwhile. The VM starts as soon as a suitable node appears, including after the node returns to service with `d8 k uncordon`.

The VM owner sees the same information in the `EvictionRequired` condition of the VM status. While the node is only being prepared for maintenance, the condition is a warning. Once eviction starts, the condition shows what happens to the VM: a move by live migration, a restart by DP, or a wait if the restart isn’t approved.

Shutting down and rebooting a node with virtual machines

Running virtual machines postpone the shutdown and reboot of their node. DP labels their workloads with `pod.deckhouse.io/inhibit-node-shutdown` and uses this label to delay the node shutdown. The mechanism is available in commercial DP editions, is described in the `node-manager` module documentation, and doesn’t need to be enabled.

If a shutdown or reboot is requested on a node that still runs virtual machines:

- The node shutdown is postponed for up to three days.
- A message about the workloads holding the shutdown is periodically printed to the node console.

On nodes where the delay mechanism works, the `GracefulShutdownPostpone` condition is always present. While the mechanism waits for a shutdown signal or holds the node, the condition has the `True` status, and once there’s nothing left to hold, it switches to `False`. What actually happens to the node is shown by the reason in the `reason` field of this condition:

- `WaitingForShutdownSignal`: The mechanism is active and waiting for a node shutdown request.
- `PodsWithLabelAreRunningOnNode`: A node shutdown is requested and postponed, because virtual machines are still running on the node.
- `NoRunningPodsWithLabel`: No virtual machines are left on the node, there’s nothing to hold the shutdown with, and it continues.

To check the reason, run the following command:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?
(@.type=="GracefulShutdownPostpone")]}{.reason}{"\n"}{end}'
```

The shutdown delay doesn’t move virtual machines to other nodes, it only keeps the node from shutting down. For this reason, free the node from virtual machines before any work that requires a shutdown or a reboot:

- If the VMs can be migrated, that is, the `Migratable` condition has the `True` status, switch the

node to maintenance mode with `d8 k drain`.

- If a VM can't be migrated, that is, the `Migratable` condition has the `False` status because of local disks or devices passed through from the node, stop it with `d8 v stop <VM_NAME>`, and start it with `d8 v start <VM_NAME>` once the work is done.

  Stopping is available only for the `Manual` and `AlwaysOnUnlessStoppedManually` run policies. Check the VM policy:

  ```
  d8 k -n <NAMESPACE> get vm <VM_NAME> -o jsonpath='{.spec.runPolicy}'
  ```

  With the `AlwaysOn` policy, the stop command is rejected with the `NotApplicableForVirtualMachineRunPolicy` reason. In that case, change the policy first, and restore the previous value once the work is done:

  ```
  d8 k -n <NAMESPACE> patch vm <VM_NAME> --type merge -p '{"spec":
  {"runPolicy":"AlwaysOnUnlessStoppedManually"}}'
  ```

  Where `<NAMESPACE>` is the project namespace, and `<VM_NAME>` is the virtual machine name.

Instead of stopping VMs manually, you can let DP restart such VMs for the duration of the node maintenance. The run policy doesn't have to be changed in that case.

If you do none of this, the node doesn't shut down. Two alerts report this situation. The `D8VirtualizationVirtualMachineHoldsNodeMaintenance` alert lists the VMs that hold the node and wait for an administrator's decision. The `D8VirtualizationNodeEvacuationStuck` alert fires if a VM was evicted from a node but neither migrated nor restarted within 15 minutes.

Additional resources

- Module `node-manager` documentation
- Module `sdn` documentation
- Module `virtualization` custom resources
- Module `virtualization` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Application scaling and resource management

Scaling applications and managing pod placement in Deckhouse Platform allows dynamic adaptation of cluster resources to the current load, ensuring high availability of services and efficient resource utilization.

Deckhouse supports all major Kubernetes scaling and workload distribution mechanisms:

- Horizontal Pod Autoscaling (HPA) — automatic adjustment of the number of pod replicas based on resource consumption metrics or external indicators.
- Vertical Pod Autoscaling (VPA) — automatic tuning of requested CPU and memory resources for containers based on actual usage.
- Scaling by metrics — using arbitrary metrics to flexibly manage application scaling through Prometheus.
- Pod redistribution (Descheduler) — automatically evicting pods to optimize workload placement across the cluster.
- Pod priorities (Priority Classes) — managing the eviction order of pods during resource shortages based on their importance.
- Scheduler — configuring the rules and logic for node selection when placing pods.

Deckhouse Platform enables automatic application scaling and efficient resource management to ensure stable and predictable cluster operation.

The following sections provide a detailed description of scaling capabilities, configuration examples, and recommendations for their effective use.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Scaling by metrics

Scaling based on metrics

Metric-based scaling is the process of automatically or manually adjusting resources (e.g., number of pod replicas, allocated CPU/memory) based on defined metrics. These metrics can range from CPU or memory usage to the number of requests per second or the size of a message queue.

Scaling can be based on both standard and custom monitoring metrics. For example, scaling can be triggered by the following metrics:

- Pod CPU — current CPU usage.
- Pod Memory — current memory usage.
- RPS (Ingress) — number of requests per second over 1, 5, or 15 minutes (`rps_1m`, `rps_5m`, `rps_15m`).
- Average CPU usage (Pod) — over 1, 5, or 15 minutes (`cpu_1m`, `cpu_5m`, `cpu_15m`).
- Average memory usage (Pod) — over 1, 5, or 15 minutes (`memory_1m`, `memory_5m`, `memory_15m`).

Configuration

To simplify metric-based scaling configuration, DP offers special resources (`Cluster*Metric` and `*Metric`). You can also configure scaling based on custom metric calculation rules.

To configure scaling based on metrics, follow these steps:

1. Enable the `prometheus-metrics-adapter` module. It allows you to define custom resources that describe metric calculation rules using PromQL queries.

   You can enable the `prometheus-metrics-adapter` module in the Deckhouse web interface or via the following command:

   ```
   d8 system module enable prometheus-metrics-adapter
   ```

2. Define a PromQL query that describes the desired metric (e.g., application request rate or CPU usage over time). This query is registered in the cluster as a metric for HPA.
3. Create a HorizontalPodAutoscaler) resource, specifying the metric name and threshold value. Kubernetes will query the result of your PromQL expression and compare it to the threshold, adjusting the number of replicas as needed (see the HPA configuration documentation for more details).

Example PromQL query:

```
query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by
(<<.GroupBy>>) OR on() vector(0)
```

In this example:

- `ingress_nginx_detail_requests_total` — the base metric (a request counter, e.g., from the Ingress NGINX controller);
- `rate(...[2m])` — calculates the rate of increase over the past 2 minutes (requests per second);
- `sum(...) by (<<.GroupBy>>)` — aggregates the results by a set of labels (e.g., Ingress name, namespace, etc.);
- `OR on() vector(0)` — a PromQL construct that ensures the result is `0` when there's no data (otherwise, the query might return nothing, which would be ignored).

This PromQL query effectively provides the current requests per second value filtered by specified labels (Ingress, namespace, etc.).

Once an administrator defines a metric using a PromQL query (e.g., in an IngressMetric object or a similar mechanism), it can be referenced in a HorizontalPodAutoscaler (HPA) as shown below:

```
apiVersion: deckhouse.io/v1beta1
kind: IngressMetric
metadata:
  name: mymetric
  namespace: mynamespace
spec:
  query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by
(<<.GroupBy>>) OR on() vector(0)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  # Metrics used for scaling.
  metrics:
  - type: Object
    object:
      # The object that owns the metrics in Prometheus.
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: myingress
      metric:
        # The metric registered via a custom resource like IngressMetric or
ClusterIngressMetric.
        # You can also use built-in metrics like rps_1m, rps_5m, or rps_15m provided
by the prometheus-metrics-adapter module.
        name: mymetric
      target:
        # For Object-type metrics, use either `Value` or `AverageValue`.
        type: AverageValue
        # Scaling will occur if the average value of the metric across all pods in the
Deployment differs significantly from 10.
        averageValue: 10
```

i When the metric exceeds (or falls below) the threshold `averageValue: 10`, the system will adjust the number of replicas of `myapp` within the range of 1 to 2.

Configuring metrics using observability module resources

To configure custom metric calculation rules:

1. Make sure the `observability` module is enabled.
2. Create a ClusterObservabilityMetricsRulesGroup or ObservabilityMetricsRulesGroup resource:
   - ClusterObservabilityMetricsRulesGroup: Intended for cluster-wide rules applied to system namespaces.
   - ObservabilityMetricsRulesGroup: Intended for rules that operate within a specific namespace.

Configuration example:

```
apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: custom-cluster-metrics-mymetric
spec:
  interval: 1m
  rules:
    - record: kube_adapter_metric_ingress_bytes_sent
      expr: |
        sum(ingress_nginx_detail_sent_bytes_sum) by (namespace, ingress)
```

Working with unstable metrics

When dealing with unstable metrics (e.g., metrics that fluctuate and cause excessive scaling), it is recommended to:

- Use PromQL aggregation functions. For example, `avg_over_time()` smooths the metric over a specified time range, helping to avoid sudden spikes:

  ```
  apiVersion: deckhouse.io/v1beta1
  kind: ServiceMetric
  metadata:
    name: rmq-queue-forum-messages
    namespace: mynamespace
  spec:
    query: sum
  (avg_over_time(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_me
  ssage",vhost="/"}[5m])) by (<<.GroupBy>>)
  ```

- Configure stabilization behavior in the autoscaler. You can increase the stabilization window so that decisions to scale up or down are based on more stable data.

Retrieving metric values

To get a list of available metrics, use the following command:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/
```

To retrieve metric values associated with specific objects, use:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-
namespace/services/*/my-service-metric
```

To get values of metrics created via `NamespaceMetric`, use:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-
namespace/metrics/my-ns-metric
```

To retrieve external metrics, use:

```
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1
```

Configuring autoscaling in the Deckhouse Web Interface

In Deckhouse Platform, you can configure node autoscaling settings through the Deckhouse web interface. This allows you to dynamically change the number of nodes depending on the load.

To configure autoscaling in the web interface:

- Open the Deckhouse web interface;
- In the left menu, go to the “Nodes” → “ Node Groups” section;
- Select the desired NodeGroup;
- In the “Autoscaling Parameters” section, the current settings will be displayed.

The following settings are available in the Deckhouse web interface:

- Nodes per zone – sets the minimum and maximum number of nodes that can run in a single zone.
- Desired – the current number of nodes needed for operation.
- Requested – the number of nodes currently scheduled for provisioning.
- Standby – additional reserved nodes.

To modify autoscaling parameters:

- Click the edit icon next to the “Nodes per zone” parameter.
- In the pop-up window, specify the minimum and maximum number of nodes.
- Click “Apply” to save the changes, or “Cancel” to discard them.

When you click “Create based on” in the web interface, a form for configuring autoscaling parameters opens:

1. **Main parameters:**
   - Zones – selection of availability zones where machines will be created. This helps distribute nodes across multiple zones. The default value depends on the selected cloud provider and usually includes all zones in the region.
   - Scaling priority – used if multiple node groups exist with the same priority. If not specified, the system will randomly choose a group for scaling.
   - Number of machines created simultaneously during scale-up – defines how many machines can be added at once during scale-up. For example, if the value is 1, nodes will be added one at a time.
   - Max unavailable instances during RollingUpdate – how many nodes can be unavailable during updates. A value of 0 means rolling updates will occur one by one with no downtime.
2. **Standby nodes and resources:**
   - Number of standby nodes out of total nodes – you can specify an exact number (e.g., 2–6) or a percentage (e.g., 15%). A standby node is a pre-provisioned node ready to receive workloads immediately. This speeds up scale-up by several minutes.
   - Resources reserved on standby nodes – the percentage of resources to reserve on standby nodes. Acceptable values: 1% to 80%. For example, if set to 50%, half of the node’s resources will be reserved.
3. **Additional parameters:**
   - Quick Shutdown – enables fast node shutdown, reducing the drain wait time to 5 minutes.

Additional resources

- Module `console` documentation
- Module `observability` documentation
- Module `prometheus-metrics-adapter` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Horizontal pod autoscaling

How Horizontal Scaling (HPA) works

Horizontal Pod Autoscaler (HPA) is a mechanism for automatically adjusting (up or down) the number of pod replicas (in Deployments or StatefulSets) based on metrics retrieved via the Kubernetes API. HPA monitors application load by checking current metrics (e.g., CPU, memory, or custom Prometheus metrics), and adjusts the number of replicas as needed to maintain a desired level of performance or to optimize resource usage.

Available metric types for HPA

Horizontal scaling in DP can be based on any available metrics, such as:

1. Pod CPU and memory usage.
   - Configured using the HorizontalPodAutoscaler resource.
     For example, you can define a Resource-type metric with `averageUtilization = 70` for CPU, so that the application scales up when average CPU usage reaches 70%.
2. DP object metric (Ingress, Service) or pod-based metrics (sum or average across all pods of a controller).

    - Enables scaling based on metrics attached to DP objects (e.g., Ingress, Service), or metrics aggregated from pods (sum or average per controller). Resources like ServiceMetric and IngressMetric are used for this.
3. Any other metrics, including external data (such as Amazon SQS metrics, cloud load balancers, SaaS services, etc.).
    - Useful when the metric source is outside the cluster. Metric calculation rules are configured using the ClusterObservabilityMetricsRulesGroup resource (for cluster-wide metrics) and the ObservabilityMetricsRulesGroup resource (for namespaced metrics) and require enabling the `observability` module.

HPA recommendations

If metrics fluctuate, wrap the metric in an aggregation function (e.g., `avg_over_time()`) or increase the stabilization window (`spec.behavior.scaleUp.stabilizationWindowSeconds`) to avoid rapid scaling of pods.

HPA limitations

1. By default, HPA handles scaling up and scaling down differently:
    - Scaling up:
        - Happens immediately (`spec.behavior.scaleUp.stabilizationWindowSeconds = 0`).
        - Limit — within 15 seconds, the number of pods can double. If there were fewer than 4 pods, up to 4 new pods can be added.
    - Scaling down:
        - Takes up to 5 minutes (`spec.behavior.scaleDown.stabilizationWindowSeconds = 300`).
        - Multiple “proposals” for a new number of replicas are collected, and the highest one is selected to avoid frequent downsizing.
        - There are no limits on how many pods can be removed at once.
2. Only one HPA per controller. You cannot assign multiple HPAs to the same Deployment (or StatefulSet) — they will conflict.

How to enable or disable HPA

HPA does not require separate activation in DP. However, if you want to scale based on metrics other than CPU and memory, you must enable the `prometheus-metrics-adapter` module. See how to enable it in the documentation.

HPA configuration

To configure HPA, follow these steps:

1. Define the controller (Deployment or StatefulSet) to scale.
2. Set scaling limits (`minReplicas` and `maxReplicas`):

   ```
   minReplicas: 1
   maxReplicas: 10
   ```

3. Configure metrics:

   ```
   metrics:
   - type: Resource
     resource:
       name: cpu
       target:
         type: Utilization
         averageUtilization: 70
   ```

4. Optionally, set the `stabilizationWindowSeconds` to delay scaling decisions and limit the pod growth rate:

   Example for scaling up:

```
behavior:
 scaleUp:
   stabilizationWindowSeconds: 300
```

Example for scaling down:

```
behavior:
 scaleDown:
   stabilizationWindowSeconds: 300
```

HPA configuration examples

Scaling based on CPU and memory

Scaling occurs when the average CPU or memory usage across all pods exceeds the specified percentage:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: app-prod
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  # Scaling boundaries for the controller.
  minReplicas: 1
  maxReplicas: 10
  # If the application tends to have short CPU usage spikes,
  # you can delay the scaling decision to confirm it's necessary.
  # By default, scaling up happens immediately.
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 300
  metrics:
  # CPU and memory-based scaling.
  - type: Resource
    resource:
      name: cpu
      target:
        # Scaling happens when the average CPU utilization across all pods in
scaleTargetRef exceeds this value.
        # For metrics of type: Resource, only target type: Utilization is available.
        type: Utilization
        # Example: if each pod requests 1 core, scaling occurs when average usage
exceeds 700m.
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        # Scaling based on memory usage exceeding a certain percentage.
        type: Utilization
        # Example: if each pod requests 1 GiB, scaling occurs when average usage
exceeds 800 MiB.
        averageUtilization: 80
```

Scaling based on object metrics

If the `rmq-queue-forum-messages` metric (number of messages in RabbitMQ) exceeds 42, the HPA increases the number of replicas in the `myconsumer` Deployment:

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"})
by (<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42
```

Scaling based on external metrics

The `mymetric` metric is sourced from an external system (e.g., SQS). Scaling is triggered when the metric value exceeds 100:

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  metrics:
  # Using external metrics for scaling.
  - type: External
    external:
      metric:
        # This is the metric registered in Prometheus as kube_adapter_metric_mymetric,
        # but used here without the 'kube_adapter_metric_' prefix.
        name: mymetric
        selector:
          # Use label selectors to narrow down the metric query.
          matchLabels:
            namespace: mynamespace
            ingress: myingress
      target:
        # Only `type: Value` is supported for External metrics.
        type: Value
        # Scaling is triggered if the metric value exceeds 10.
        value: 10
```

Additional resources

- Module `observability` documentation
- Module `prometheus-metrics-adapter` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Vertical pod autoscaling

How Vertical Scaling (VPA) works

Vertical Pod Autoscaler (VPA) automates container resource management and significantly improves application efficiency. VPA is useful in scenarios where the exact resource requirements of an application are unknown in advance. When VPA is enabled and the appropriate operating mode is set, the requested resources are determined based on actual usage metrics gathered from the monitoring system. It is also possible to configure the system to only recommend resource settings without applying them automatically.

If the application load changes depending on time of day, user requests, or other factors, VPA automatically adjusts the allocated resources. This helps prevent outages due to lack of resources or excessive CPU and memory consumption.

For details on VPA operating modes, its components and limitations, refer to Architecture.

How to enable or disable VPA

You can enable VPA in the following ways:

1. Via a ModuleConfig resource (e.g., ModuleConfig/vertical-pod-autoscaler). Set the `spec.enabled` parameter to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: vertical-pod-autoscaler
   spec:
     enabled: true
   ```

2. Via the `d8` command (run inside the `d8-system/deckhouse pod`):

   ```
   d8 system module enable vertical-pod-autoscaler
   ```

3. Through the Deckhouse web interface:

   - Go to the “Deckhouse → Modules” section;
   - Find the vertical-pod-autoscaler module and click on it;
   - Toggle the “Module enabled” switch.

The `vertical-pod-autoscaler` module has no required configuration — it can be enabled and used with default settings.

After creating a VerticalPodAutoscaler resource, you can check VPA recommendations using the following command:

```
d8 k describe vpa my-app-vpa
```

In the `status` section, you’ll see the following parameters:

- `Target`: The recommended resource amount for the pod.
- `Lower Bound`: The minimum recommended resource amount for the application.
- `Upper Bound`: The maximum recommended resource amount for the application.
- `Uncapped Target`: The resource value based on extreme metrics without considering historical data.

VPA configuration

1. Create the VPA module configuration.

   To configure VPA, you need to create a configuration file for the module. Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: vertical-pod-autoscaler
   spec:
     version: 1
     enabled: true
     settings:
       nodeSelector:
         node-role/system: ""
       tolerations:
       - key: dedicated.deckhouse.io
         operator: Equal
         value: system
   ```

2. Apply the VPA configuration file using `d8 k apply -f <your-config-file-name>`.

For more details on VPA resource limits configuration, see the documentation.

Module configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
```

Additional resources

- Module `console` documentation
- Module `vertical-pod-autoscaler` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Pod restart on configuration change

Deckhouse Platform can automatically restart Pods when certain ConfigMap or Secret resources are modified. This functionality is based on the Reloader project and is controlled via annotations added to Pod controllers (Deployment, DaemonSet, StatefulSet).

> Reloader is not designed to be highly available.

The following annotations allow you to control pod restarts.

Supported annotations

Annotation	Applies to objects	Purpose	Example values
`pod-reloader.deckhouse.io/auto`	Deployment, DaemonSet, StatefulSet	Automatically restarts Pods when any related ConfigMap or Secret changes (used as a volume or environment variable)	`"true"`, `"false"`
`pod-reloader.deckhouse.io/search`	Deployment, DaemonSet, StatefulSet	Restarts only when resources annotated with `pod-reloader.deckhouse.io/match: "true"` are changed	`"true"`, `"false"`
`pod-reloader.deckhouse.io/configmap-reload`	Deployment, DaemonSet, StatefulSet	Specifies particular `ConfigMap` resources that should trigger a restart when changed	`"some-cm"`, `"some-cm1,some-cm2"`
`pod-reloader.deckhouse.io/secret-reload`	Deployment, DaemonSet, StatefulSet	Specifies particular `Secret` resources that should trigger a restart when changed	`"some-secret"`, `"some-secret1,some-secret2"`

Annotation	Applies to objects	Purpose	Example values
`pod-reloader.deckhouse.io/match`	ConfigMap, Secret	Marks resources to be tracked when `pod-reloader.deckhouse.io/search: "true"` is used	`"true"`, `"false"`

⚠ The `pod-reloader.deckhouse.io/search` annotation must not be used together with `pod-reloader.deckhouse.io/auto: "true"`. In this case, both `pod-reloader.deckhouse.io/search` and `pod-reloader.deckhouse.io/match` annotations will be ignored. To ensure correct behavior, set `pod-reloader.deckhouse.io/auto: "false"` or remove it.

The `pod-reloader.deckhouse.io/configmap-reload` and `pod-reloader.deckhouse.io/secret-reload` annotations do not work when `pod-reloader.deckhouse.io/auto: "true"` is present. To ensure correct behavior, disable `auto`.

Usage examples

Tracking all changes in all attached resources

Connected resources can be used either as environment variables or mounted as volumes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
  annotations:
    pod-reloader.deckhouse.io/auto: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages
```

Tracking changes in specific resources

Indicates that only those Secrets or ConfigMaps with the annotation `pod-reloader.deckhouse.io/match: "true"` should be monitored. This is useful when a pod uses many resources, but restarts are required only upon changes to specific ones.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/search: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
  annotations:
    pod-reloader.deckhouse.io/match: "true"
```

Tracking changes in resources from the list

Explicitly specifies a list of ConfigMaps that should trigger a pod restart when changed.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/configmap-reload: "nginx-config,nginx-pages"
spec:
  template:
    spec:
      containers:
        - name: nginx
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
            - name: config
              mountPath: "/etc/nginx/templates"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
        - name: config
          configMap:
            name: nginx-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
```

Enabling or disabling pod restart

You can enable or disable the pod restart functionality in the following ways:

1. Using a `ModuleConfig` resource (e.g., `ModuleConfig/pod-reloader`). Set the `spec.enabled` field to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: pod-reloader
   spec:
     enabled: true
   ```

2. Using the `d8` command (in the `d8-system/deckhouse` pod):

   ```
   d8 system module enable pod-reloader
   ```

3. Through the Deckhouse web interface:
   - Go to the “Deckhouse → Modules” section;
   - Find the `pod-reloader` module and click on it;
   - Toggle the “Module enabled” switch.

Configuration

The pod restart mechanism works out of the box and does not require any mandatory configuration. By default, it is enabled in the Default and Managed module bundles and disabled in the Minimal bundle.

If needed, its behavior can be adjusted in the settings of the pod-reloader module (ModuleConfig `pod-reloader`).

Available parameters:

Parameter	Type	Description	Default Value
`reloadOnCreate`	boolean	Restart on ConfigMap or Secret creation, not only on modification	`true`
`nodeSelector`	object	Limits the nodes where the component can run (equivalent to `spec.nodeSelector`)	Not set
`tolerations`	array	Allows scheduling on tainted nodes (equivalent to `spec.tolerations`)	Not set

Additional resources

- Module `console` documentation
- Module `pod-reloader` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Scheduler

Pod scheduling

General overview of the scheduler

The scheduler is a component responsible for assigning pods to cluster nodes based on available resources and specified rules. It selects the most suitable node for running each pod, considering many factors such as CPU and memory availability, data locality, network topology, node labels, and application-specific requirements.

The scheduler operates in multiple stages, each involving different plugins. These plugins evaluate nodes against specific criteria and help determine the optimal placement for a pod.

Main tasks of the scheduler:

- Filtering — determines which nodes meet the pod’s requirements.
- Scoring — assigns scores to nodes based on various criteria (the higher the score, the better the node).
- Binding — assigns the pod to the best node based on scoring results.

Plugins may participate in one or multiple stages. For example, one plugin might be used only during filtering, while another can be involved in both filtering and scoring.

Examples of plugins:

- ImageLocality (phase: Scoring) — prefers nodes that already have the required container images. This helps reduce deployment time since images do not need to be pulled from a remote repository.
- TaintToleration (phases: Filtering, Scoring) — implements the taints and tolerations mechanism. It helps avoid placing pods on unsuitable nodes or, conversely, forces placement on specific nodes (e.g., those with dedicated resources).
- NodePorts (phase: Filtering) — checks if the node has free ports required by the pod. This is especially important for services using fixed NodePorts, as a node cannot host multiple pods using the same port.

For the complete list of plugins, refer to the Kubernetes documentation.

Pod scheduling phases

The process of assigning pods to nodes in DP involves several key phases. The scheduler analyzes available nodes, applies various filtering and scoring criteria, and then selects the most suitable node to run the pod.

In the Filtering phase, filtering plugins are activated. These plugins evaluate all available nodes and select only those that meet certain conditions (e.g., taints, nodePorts, nodeName, unschedulable, and others).

If the nodes are distributed across different availability zones, DP alternates between zones during selection to avoid placing all pods in a single zone. For example, if the nodes are distributed like this:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

The selection will occur in the following order:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4
```

It is important to note that the scheduler does not evaluate all nodes in the cluster — only a subset is considered to optimize the scheduling process. The number of nodes evaluated depends on the size of the cluster:

- In small clusters (≤50 nodes), all nodes are evaluated.
- In medium clusters (~100 nodes), about 50% of nodes are evaluated.
- In very large clusters (~5000 nodes), only 10% of nodes are evaluated.
- The minimum threshold is 5% of nodes when there are more than 5000.

After the filtering phase, the scoring phase begins. During this stage, each node from the filtered list is scored by various scoring plugins that analyze its characteristics and how well it matches the pod's requirements.

Each plugin evaluates a node based on its specific criteria and assigns it a score. These scores are then aggregated to produce an overall ranking.

Key factors considered during scoring include:

- Node load (`pod capacity`) — available CPU and memory are taken into account. Nodes with more free resources receive higher scores.
- Container image locality (ImageLocality) — nodes that already have the required container images are preferred, as this speeds up pod deployment.
- Affinity and anti-affinity rules — nodes that satisfy required placement (or avoidance) rules based on other pods get higher priority.
- Storage availability (`volume provisioning`) — nodes with the necessary PersistentVolumes receive a higher score.

After all scores are summed, the node with the highest total score is selected for pod placement.

If multiple nodes receive the same highest score, the selection is made randomly to avoid inefficient and predictable load distribution.

How to modify or extend scheduler logic

The scheduler in DP can be flexibly configured and extended using custom plugins. This allows its behavior to be adapted to specific cluster requirements, such as considering custom metrics, special workload distribution rules, or node prioritization.

Each such plugin is implemented as a webhook and must meet the following requirements:

- TLS usage – The plugin must be accessed over a secure TLS connection.
- Availability – The plugin must be deployed as a service inside the cluster and accessible via HTTP(S).
- Support for standard verbs – The plugin must support standard scheduler operations:
  - `filterVerb = filter` – participates in node filtering before pod placement.
  - `prioritizeVerb = prioritize` – participates in node scoring during the Scoring phase.
- Node cache capability – It is expected that all extender plugins support caching node data (`nodeCacheCapable: true`) to improve performance.

To connect an extender plugin, use the KubeSchedulerWebhookConfiguration resource. This resource

defines the configuration for an external webhook used by the kube-scheduler and enables more advanced scheduling conditions in the cluster, such as:

- Placing application pods closer to data storage nodes,
- Prioritizing nodes based on their current state (e.g., network load or storage subsystem health),
- Segmenting nodes into zones, etc.

Example of connecting an external scheduler plugin via webhook:

```
apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
  timeout: 5s
```

✖ If the `failurePolicy: Fail` parameter is used, any failure in the webhook will cause the scheduler to stop functioning entirely, preventing new pods from being scheduled. It is highly recommended to thoroughly test any custom scheduler plugins before deploying them in a production environment.

Additional pod placement mechanisms

Deckhouse Platform provides flexible mechanisms for managing pod placement within a cluster. These mechanisms help optimize load balancing, improve fault tolerance, and separate system and user workloads.

Main placement configuration options:

- Using node labels (`NodeGroup.spec.nodeTemplate.labels`).
  Allows explicitly targeting specific nodes for pods using `spec.nodeSelector` or `spec.affinity.nodeAffinity`.
- Configuring taints and tolerations.
  Enables limiting pod scheduling to specific nodes, preventing them from being placed on unsuitable hosts.
- Using custom toleration keys (`settings.modules.placement.customTolerationKeys`).
  Allows control over the scheduling of critical Deckhouse components (e.g., CNI and CSI) on dedicated nodes.

1. Using `nodeSelector` — allows you to explicitly specify which nodes pods should be scheduled on. This is done by labeling the desired nodes and referencing those labels in the pod's `spec.nodeSelector`. Example:

   Suppose you have a node named `kube-system-1` that is intended for monitoring services. Label the node:

   ```
   d8 k label node kube-system-1 node-role/monitoring=""
   ```

   Now, to deploy pods only on this node, add a `nodeSelector` to the Deployment:

   ```
   nodeSelector:
    node-role/monitoring: ""
   ```

2. Using taints and tolerations — unlike `nodeSelector`, taints are used to prevent pods from being scheduled on a node unless they have a matching toleration. This ensures that only specific

services run on a given node. Example:

Suppose you have a node named `kube-frontend-1` that is dedicated exclusively to Ingress controllers. Apply a taint to the node:

```
d8 k taint node kube-frontend-1 node-role/frontend="":NoExecute
```

Now, only pods with the appropriate toleration will be allowed to run on this node:

```
tolerations:
- effect: NoExecute
  key: node-role/frontend
```

This mechanism prevents unintended pods from being scheduled on nodes that are meant for specific purposes.

3. Using `customTolerationKeys` — Deckhouse supports the `customTolerationKeys` mechanism, which explicitly defines allowed toleration keys. This is useful when you need to run system services (such as CNI, CSI, etc.) on specific nodes. Example:

   ```
   customTolerationKeys:
   - dedicated.example.com
   - node-dedicated.example.com/master
   ```

   This mechanism allows:

   - Divide nodes into zones — for example, dedicating some for Ingress controllers and others for system services (Prometheus, VPN, CoreDNS).
   - Isolate critical applications from system components, avoiding resource contention.

Scheduler profiles

The scheduler supports multiple profiles that define different strategies for distributing pods across nodes in the cluster. Depending on the selected profile, pods will be scheduled with different load balancing logic.

- `default-scheduler` — the default profile that attempts to evenly distribute pods across nodes, preferring less loaded ones.
- `high-node-utilization` — a profile that places pods on more heavily loaded nodes. This can be useful in scenarios where workload consolidation is needed, allowing underutilized nodes to be shut down or repurposed.

To select a scheduler profile, specify it in the `spec.schedulerName` field of the Pod manifest. Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
  - name: example-pod
    image: registry.k8s.io/pause:2.0
```

Pod redistribution

DP analyzes the state of the cluster every 15 minutes and evicts pods that match the conditions described in active scheduling strategies. Evicted pods go through the standard scheduling process again, taking the current cluster state into account. This mechanism allows workloads to be redistributed based on the selected strategy and frees up resources on certain nodes when necessary.

Considering pod Priority Classes

DP defines a set of priority classes that determine the importance of pods and the order in which they

are evicted during workload redistribution.

If the cluster lacks resources, lower-priority pods may be evicted in favor of higher-priority ones. This ensures that critical services remain operational even when nodes are overloaded.

How DP considers pod priority

Deckhouse Platform uses the pod priority mechanism to determine which pods should be evicted when resources are insufficient. The higher the pod’s priority, the lower the chance it will be evicted during redistribution.

This mechanism is controlled by the `descheduler` module, where you can define a priority threshold using the `spec.priorityClassThreshold` parameter. This threshold limits eviction to only those pods with a priority below the specified value.

You can set the threshold in two ways:

- By class name: Use `priorityClassThreshold.name` to evict only pods with a priority lower than the specified priority class.
- By numeric value: Use `priorityClassThreshold.value` to evict pods with a priority lower than the specified integer value.

Example:

```
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: custom
spec:
  priorityClassThreshold:
    name: high-priority
```

Which pods are not evicted

- DP does not evict a pod in the following cases:
  - The pod is in the `d8-*` or `kube-system` namespace;
  - The pod has the `priorityClassName` set to `system-cluster-critical` or `system-node-critical`;
  - The pod is using local storage;
  - The pod is managed by a DaemonSet;
  - Evicting the pod would violate a Pod Disruption Budget (PDB);
  - There are no available nodes to reschedule the evicted pod.

If multiple pods match the eviction criteria, DP applies additional logic:

1. Pods with the lowest priority (`BestEffort`) are evicted first;
2. Then `Burstable` pods are considered;
3. `Guaranteed` pods are evicted last, and only if absolutely necessary.

DP provides fine-grained control over which pods and nodes are subject to eviction:

- `spec.podLabelSelector` — limits evicted pods by label;
- `spec.namespaceLabelSelector` — filters namespaces whose pods can be considered for eviction;
- `spec.nodeLabelSelector` — selects target nodes by label.

Each of these fields supports standard `matchExpressions` and `matchLabels` syntax, allowing you to use operators like `In`, `NotIn`, `Exists`, and `DoesNotExist` with desired label values.

How to enable or disable pod eviction

To enable the pod redistribution feature, you need to enable the `descheduler` module.

You can do this in one of the following ways:

1. Using a ModuleConfig resource (e.g., `ModuleConfig/descheduler`). Set the `spec.enabled` field to `true` or `false`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true   # or false to disable
```

2. Using the `d8` command (in the `d8-system/deckhouse` pod):

```
d8 system module enable descheduler
# or to disable:
d8 system module disable descheduler
```

3. Through the Deckhouse web interface:
   - Go to the “Deckhouse → Modules” section;
   - Find the `descheduler` module and click on it;
   - Toggle the “Module enabled” switch.

The module does not require mandatory configuration. You can enable it without additional settings — it will work with the default values.

Descheduling strategies

The `spec.strategies` parameter lists the strategies you want to enable or configure. Each strategy has an `enabled` flag (default is `false`).

Below is a list of the main strategies available in DP.

HighNodeUtilization — concentrates workloads on fewer nodes by evicting pods from underutilized nodes so they can be rescheduled elsewhere. Requirements:

- Specific `descheduler` module configuration — `MostAllocated`;
- (Optional) cluster autoscaling must be enabled — to allow unused nodes to be shut down.

This strategy is enabled using the `spec.strategies.highNodeUtilization.enabled` parameter.

The `thresholds` parameter defines the resource usage levels below which a node is considered underutilized. If usage (CPU, memory, etc.) falls below *all* threshold values, the node is deemed underutilized, and DP will attempt to evict pods from it.

Example:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

If any of these resource types is not specified, all its thresholds default to 100 to avoid nodes going from underutilized to overutilized For example, If the memory threshold is not specified in thresholds, it defaults to 100.

```
thresholds:
  cpu: 50
```

> **i** In GKE (Google Kubernetes Engine), it is not possible to configure `MostAllocated` by default, but you can use the `optimize-utilization` strategy instead.

LowNodeUtilization — more evenly distributes workloads across nodes. This strategy identifies underutilized nodes and evicts pods from overutilized ones, assuming that evicted pods will be rescheduled on the underutilized nodes.

Enable the strategy via `spec.strategies.lowNodeUtilization.enabled`.

- An **underutilized** node is one where all resources are below the thresholds set in `strategies.lowNodeUtilization.thresholds`.
- An **overutilized** node exceeds at least one of the values in `strategies.lowNodeUtilization.targetThresholds`.

Nodes with resource usage between `thresholds` and `targetThresholds` are considered optimally utilized and will not have pods evicted.

Example:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

RemoveDuplicates — prevents multiple pods from the same controller (ReplicaSet/ReplicationController/StatefulSet/Job) from running on the same node simultaneously (DaemonSets are excluded). If, for any reason, 2 or more pods from the same controller end up on a single node, this strategy will evict the “extra” pods to redistribute them more evenly across the cluster.

This situation can occur after temporary node failures — when pods are rescheduled to other nodes, and once the original node is back online, the pods remain concentrated on the fallback node.

Enable the strategy using the `strategies.removeDuplicates.enabled` parameter.

Example:

```
spec:
  strategies:
    removeDuplicates:
      enabled: true
```

RemovePodsHavingTooManyRestarts — evicts pods having too many restarts from nodes. Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold. This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

Enable the strategy using the `strategies.removePodsHavingTooManyRestarts.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsHavingTooManyRestarts:
      enabled: true
```

RemovePodsViolatingInterPodAntiAffinity — evicts any pods that violate `inter-pod affinity/anti-affinity` rules. For example, if `pod2` and `pod3` have an `anti-affinity` rule against `pod1`, but all three end up on the same node, this strategy will evict `pod1` so that `pod2` and `pod3` can continue operating according to their scheduling constraints.

Enable the strategy using the `strategies.removePodsViolatingInterPodAntiAffinity.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingInterPodAntiAffinity:
      enabled: true
```

RemovePodsViolatingNodeAffinity — this strategy evicts pods that no longer comply with their defined `node affinity` rules. `Node affinity` defines which nodes are suitable for a pod at scheduling time:

1. `requiredDuringSchedulingIgnoredDuringExecution`:
   - A pod with this rule must be scheduled only on a node that satisfies the specified conditions (e.g., a specific label).
   - If the node later no longer satisfies the condition (e.g., the label is removed), and there is another suitable node in the cluster, DP will evict the pod so it can be rescheduled on a valid node.
2. `preferredDuringSchedulingIgnoredDuringExecution`:
   - A pod with this preference may run on a node that doesn’t fully match the affinity, if no better option is available.
   - If a more suitable node appears later in the cluster, DP may evict the pod so it can be restarted under better placement conditions.

This strategy helps ensure pods continue to align with their node affinity rules and do not remain in suboptimal locations when better options become available.

Enable the strategy using the `spec.strategies.removePodsViolatingNodeAffinity.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingNodeAffinity:
      enabled: true
      nodeAffinityType:
        - requiredDuringSchedulingIgnoredDuringExecution
        - preferredDuringSchedulingIgnoredDuringExecution
```

RemovePodsViolatingTopologySpreadConstraint — ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint’s `maxSkew`. This is useful for rebalancing pods across zones after a zone outage recovery.

Enable the strategy using the `strategies.removePodsViolatingTopologySpreadConstraint.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingTopologySpreadConstraint:
      enabled: true
```

Additional resources

- Global parameters
- Module `console` documentation
- Module `control-plane-manager` custom resources
- Module `descheduler` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Priority classes

Deckhouse Platform creates a set of priority classes (PriorityClass) in the cluster and assigns them to its components. Applications and virtual machines (VMs) can use these classes by setting the `priorityClassName` field. The scheduler takes pod priority into account: if resources are insufficient, pods with lower priority are preempted first.

For example, if a pod with the `production-low` class cannot be scheduled due to insufficient resources, the scheduler first preempts pods with a lower priority, such as `develop`, then `cluster-low`, and so on. If `priorityClassName` is not set, the pod is assigned the `develop` class.

The priority class of a VM is passed to its `virt-launcher` pod, so the scheduler uses the same rules to choose which VMs to preempt as it does for pods.

Available priority classes

The table lists the priority classes that Deckhouse Platform creates in the cluster, in descending priority order (the higher the value, the higher the priority). Choose a class based on the environment and workload criticality.

> ✖ Do not use the `system-node-critical`, `system-cluster-critical`, `cluster-medium`, or `cluster-low` priority classes, as they are reserved for critical cluster components.

Priority class	Description	Value
`system-node-critical`	Cluster components that must be present on the node. Also fully protected from eviction by the kubelet. Examples: `node-exporter`, `csi`, and others.	2000001000
`system-cluster-critical`	Cluster components without which the cluster cannot operate correctly. MutatingWebhooks and Extension API servers must use this priority class. Also fully protected from eviction by the kubelet. Examples: `kube-dns`, `kube-proxy`, `cni-flannel`, `cni-cilium`, and others.	2000000000
`production-high`	Stateful applications whose absence in a production environment leads to complete service unavailability or data loss. Examples: `PostgreSQL`, `Memcached`, `Redis`, `MongoDB`, and others.	9000

Priority class	Description	Value
`cluster-medium`	Cluster components that affect monitoring (alerts, diagnostics) and autoscaling. Without monitoring, you cannot assess the scope of an incident; without autoscaling, applications cannot get the resources they need. Examples: `deckhouse`, `node-local-dns`, `grafana`, `upmeter`, and others.	7000
`production-medium`	Main stateless applications in production that serve end users.	6000
`deployment-machinery`	Cluster components used for building and deploying workloads to the cluster.	5000
`production-low`	Production applications (cron jobs, admin panels, batch processes) that can be unavailable for some time. If batch or cron jobs must not be interrupted, assign them `production-medium`.	4000
`staging`	Staging environments for applications.	3000
`cluster-low`	Cluster components that are not required for operation but are desirable. Examples: `dashboard`, `cert-manager`, `prometheus`, and others.	2000
`develop` (default)	Development environments for applications. Default class if no other class is specified.	1000
`standby`	Not intended for applications. Used for system purposes to reserve nodes.	-1

Creating additional priority classes

In addition to the classes that Deckhouse Platform creates in the cluster, you can create an additional priority class by specifying a name and a numeric priority value.

The following example creates the `critical-applications` priority class with priority `8000`. Pods that use this priority class are preempted before pods with the `production-high` priority class (and higher), but after pods with the `cluster-medium` priority class (and lower).

Create a file `critical-applications.yaml` with the following content:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: critical-applications
value: 8000
globalDefault: false
description: "Priority for critical applications"
```

A PriorityClass resource manifest uses the following fields:

- `metadata.name` — the priority class name referenced by the `priorityClassName` field in pod specifications.
- `value` — numeric priority. The higher the value, the higher the priority.
- `globalDefault` — specifies whether this class is assigned to pods that do not explicitly set

`priorityClassName`.
- `description` — class description.

⚠ Be careful with the `globalDefault: true` field. If you enable it for an additional class, that class will be assigned to pods that do not explicitly specify `priorityClassName`, which may cause unexpected preemption of system components.

Apply the manifest to the cluster:

```
d8 k apply -f critical-applications.yaml
```

Verify that the priority class was created:

```
d8 k get priorityclass critical-applications
```

Example output:

```
NAME                    VALUE   GLOBAL-DEFAULT   AGE   PREEMPTIONPOLICY
critical-applications   8000    false            7s    PreemptLowerPriority
```

i Do not create additional classes with values above 1,000,000 to avoid disrupting critical system components.

Preemption mechanism

How it works

The scheduler makes preemption decisions based on numeric priority values. The workload type (Deployment, StatefulSet, DaemonSet, VM, or standalone pod) does not affect preemption decisions. The scheduler compares pod priority values regardless of the workload type: a higher-priority pod can preempt a lower-priority pod to free resources on the node.

Preemption does not occur when priorities are equal. If a new pod has the same priority as existing pods, the scheduler does not preempt them. The new pod remains `Pending` until sufficient resources become available.

For practical scenarios and step-by-step demonstrations, see Using priority classes.

Protecting a pod from eviction

The only way to protect a pod from preemption is to assign it a sufficiently high priority. However, even with a high priority, the pod will be preempted if another pod with an even higher priority appears in the cluster.

To reduce the risks associated with preemption, combine the following three mechanisms:

1. Specifying a high-priority class in the `priorityClassName` field — reduces the likelihood that the pod will be selected for preemption.
2. Using a PodDisruptionBudget (PDB) — limits how many replicas can be disrupted at the same time. During preemption, the scheduler treats PDB constraints as best-effort rather than strict requirements (unlike planned operations such as draining a node, where PDB is a hard constraint).
3. `terminationGracePeriodSeconds` — gives the application time to flush buffers and close connections before forced termination. This helps preserve data integrity when a pod must be terminated, including cases where the PDB cannot be honored.

i Without the `terminationGracePeriodSeconds` parameter on a pod, an immediate termination may lead to loss of data that was not written to disk and, for example, to database corruption.

Do not create a PodDisruptionBudget for VMs: it does not affect VM eviction. The time for a graceful guest operating system (OS) shutdown is set by the `.spec.terminationGracePeriodSeconds` parameter in the VM specification (60 seconds by default).

Differences between priority classes and other resource management

mechanisms

Priority classes (PriorityClass) serve a different purpose from other resource management mechanisms:

Mechanism	Purpose	Difference from priority classes
ResourceQuota	Limits resource consumption in a namespace	Limits total resource consumption within a namespace; does not determine preemption order.
LimitRange	Default limits and requests for pods	Sets minimum and maximum values; does not affect preemption.

Unlike ResourceQuota and LimitRange, priority classes determine which pods take precedence when resources are insufficient; they do not limit resource consumption.

Operations and diagnostics

If a pod does not start despite available resources in the cluster, the issue is often related to pod placement rather than its priority class.

Possible cluster-level causes:

- Available CPU and memory are distributed across multiple nodes, and no single node has enough resources to accommodate the pod, even after preemption.
- No eligible pods can be preempted because all pods on the target node have equal or higher priority. How to check — see the Practical check when no suitable pods are available user section.
- No nodes satisfy the pod's scheduling requirements, for example because of taints that the pod does not tolerate.
- The pod limit per node (`Capacity.pods`) has been reached. Preemption cannot resolve this condition because replacing one pod with another does not reduce the total pod count.

What you can do:

1. Add worker nodes or redistribute workloads if available resources are spread across different nodes.
2. Free resources on the overloaded node by removing or relocating less critical pods.
3. Review the density of DaemonSet and system pods on the node.
4. If needed, assign the application a higher priority class from Available priority classes.

Additional resources

- Using priority classes

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Network

Unlike basic Kubernetes installations, Deckhouse Platform (DP) handles network configuration through modules and custom resources, which simplifies and standardizes the setup process.

The platform supports:

- Management of incoming and outgoing traffic.
- Configuration of inter-cluster communication.
- Management of network policies and routing.
- DNS management within the Kubernetes cluster.

Configuring and managing incoming traffic

Incoming traffic is managed using Ingress controllers. These controllers route user requests to the appropriate applications and services based on rules defined in Ingress resources. In DP, traffic can be

managed at both the network level (NLB – Network Load Balancer) and the application level (ALB – Application Load Balancer).

Managing outgoing traffic

DP administrators can configure and manage outgoing traffic to ensure correct processing and routing of all outgoing data.

Configuring outgoing traffic enables connection control and filtering, which is especially important for secure integrations with external APIs and services. This can be achieved by defining policies that allow or block specific traffic flows.

Outgoing traffic management is implemented using the Egress Gateway feature (provided by the `cni-cilium` module) as well as Istio tools (via the `istio` module).

Internal network configuration

DP provides a wide range of options for managing the internal network. Administrators can configure pod-to-pod and pod-to-node communication, as well as traffic encryption using various technologies and tools.

Configuring inter-cluster communication

DP supports two approaches to organizing communication between independent and codependent clusters:

- Multicluster — combines clusters to share resources, balance loads, and improve fault tolerance. Ideal for distributed teams and infrastructure spread across regions.
- Federation — centralized management of multiple independent clusters with unified policies and synchronized configurations. Effective for large organizations and complex systems.

For more information on configuring inter-cluster communication, refer to Inter-cluster communication.

Network policies

Network policies in DP describe rules that regulate interactions between objects within the cluster and define ingress and egress traffic for those objects. They isolate pods, protect against internal cluster attacks, and control access to external services and filter incoming and outgoing connections.

The section includes the following pages:

- Kubernetes NetworkPolicy — isolation model, selectors, default policies, API limitations.
- CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy — Cilium extensions, entities, L7 rules, FQDN rules, audit mode.
- Host firewall on nodes — protecting nodes with CiliumClusterwideNetworkPolicy and `nodeSelector`.
- Common policy examples — recipes for typical tasks.
- Diagnostics and observability — verifying and debugging policies.

Virtual machine networking

Virtual machines get their addresses from separate subnets listed in the `virtualMachineCIDRs` parameter of the virtualization settings. Which subnets to choose, how they differ from the pod and service networks, and how a machine gets its address is described in Virtual machine networking.

Additional resources

- Module `cni-cilium` documentation
- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Internal network configuration

In Deckhouse Platform, networking is configured using CNI plugins. The recommended option is Cilium, which suits most use cases. Other supported CNIs include Flannel and Simple Bridge, typically used with cloud providers.

Network parameters are specified during the deployment of the DP cluster:

- Address ranges for Pods and Services are defined.
- The network operation mode for Cilium is selected (this can be changed later).

Internal network operation modes

The operation mode is set via the `tunnelMode` parameter in the `cni-cilium` module settings. Two modes are supported:

- Classic
- VXLAN tunnel

The appropriate mode depends on the cluster network setup:

- If all nodes are on the same network, either mode can be used.
- If nodes are on different networks, using VXLAN is recommended to create a virtual isolated network over the existing one. This allows communication across different network segments.

Performance differences between the modes are minimal. The table below outlines their characteristics:

Parameter	Classic mode	VXLAN tunnel mode
Network type	For nodes in the same network	Suitable for nodes in different networks
Traffic isolation	Not provided	Provided
Infrastructure usage	Direct routing	Virtual network over existing infrastructure
Changes to MTU	No changes	MTU is reduced due to encapsulation
Additional routing	Intermediate hop is added	Not required

Mode configuration

The internal network mode is configured via the ModuleConfig resource named `cni-cilium`.

Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings:
    tunnelMode: VXLAN
```

For more details on configuring the `cni-cilium` module, refer to the module documentation.

⚠ After changing the operation mode, reboot all nodes. Otherwise, Pod availability issues may occur.

Services

Services in DP are implemented either using the selected CNI plugin or through `kube-proxy`.

They provide access to groups of Pods performing the same function and balance traffic between them. Each Service object defines:

- A logical group of endpoints (Pods).

- A routing method for reaching them.

Supported service types

- **ClusterIP**: Creates an internal IP address selected from the Service network. All traffic sent to this IP is forwarded to the Pods specified in the selector. This ensures load balancing.

  You can also disable IP assignment in this service type. In that case, the service will not be assigned an IP address from the service network, and it will resolve to the Pod IPs. Load balancing will be done using DNS (round-robin).

- **NodePort**: Opens the specified port on each cluster node and forwards incoming traffic to the Pods. For security reasons, DP only listens on the node's internal IP. This behavior can be overridden by adding the annotation `node.deckhouse.io/nodeport-bind-internal-ip: "false"` to a node group.
- **LoadBalancer**: Created in the cloud provider where DP is deployed and accepts external traffic. In bare-metal clusters, similar functionality is implemented using the `metalLb` module.
- **ExternalName**: A DNS record for accessing the service (acting as a CNAME).

Configuring the service network with kube-proxy

When using Flannel or Simple Bridge, the `kube-proxy` module is required for services to function. It replaces the default kubeadm components (DaemonSet, ConfigMap, RBAC) with its own configuration.

By default, for security reasons, when using services of NodePort type, connections are only accepted on the node's InternalIP. This behavior can be changed by adding the following annotation to the node: `node.deckhouse.io/nodeport-bind-internal-ip: "false"`.

Example annotation for a NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
```

⚠ After adding, removing, or changing this annotation, you must manually restart all kube-proxy Pods.

When the `cni-cilium` module is enabled, the `kube-proxy` module is automatically disabled.

Additional resources

- Module `cni-cilium` documentation
- Module `metallb` parameters

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Encrypting traffic between pods

To encrypt traffic between pods in Deckhouse Platform (DP), you can use mTLS provided by Istio (via the `istio` module).

mTLS (mutual TLS) enables mutual authentication between services using TLS certificates: for all outgoing requests, the server certificate is verified; for incoming requests, the client certificate is checked. Once verification is successful, the sidecar proxy identifies the remote node and can use this information for authorization or other application-level purposes.

Each service receives a unique identifier in the format `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`.

Where:

- `TrustDomain` is the cluster domain.
- `ServiceAccount` is the account name (either `default` or a custom one).

This identifier is used as the verifiable name in TLS certificates.

The settings can be overridden at the namespace level.

mTLS configuration example

1. Enable the `istio` module:

   ```
   d8 k create -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: istio
   spec:
     version: 2
     enabled: true
   EOF
   ```

2. Create a namespace and add labels:

   ```
   d8 k create namespace test-istio-mtls
   d8 k label namespace test-istio-mtls istio-injection=enabled
   d8 k label namespace test-istio-mtls security.deckhouse.io/pod-policy=privileged
   ```

3. Apply a policy that defines the mTLS mode. Available modes:

   - `PERMISSIVE` (default): Allows encrypted traffic between pods, but also permits unencrypted traffic.
   - `STRICT`: Only encrypted traffic is allowed.

   Example of applying `PERMISSIVE` mode:

   ```
   d8 k -n test-istio-mtls create -f -<<EOF
   apiVersion: security.istio.io/v1beta1
   kind: PeerAuthentication
   metadata:
     name: default
   spec:
     mtls:
       mode: PERMISSIVE
   EOF
   ```

4. Create a Deployment:

   - If using a public registry:

     ```
     d8 k -n test-istio-mtls create deployment webserver --image=docker.io/library/nginx:1.26-alpine --port 80
     ```

   - If using an all-in-one image (replace with your registry address):

     ```
     d8 k -n test-istio-mtls create deployment webserver --image=registry.company.network/localrepo/all-in-one-image:0.1 --port 80 -- /bin/sh -c 'nginx -g "daemon off;"'
     ```

5. Expose the Deployment:

```
d8 k -n test-istio-mtls expose deployment webserver --port=80
```

6. Get the Pod IP or the associated node name. It will be required to run `tcpdump` on the respective node:

   ```
   d8 k -n test-istio-mtls get pods -l app=webserver -o wide
   ```

   Example output:

   ```
   NAME                         READY   STATUS     RESTARTS   AGE   IP
   NODE                                 NOMINATED NODE   READINESS GATES
   webserver-76d6c9b8c-9mdtb    2/2     Running    0          48m   10.111.1.122
   test-worker-e36e4712-5948b-sp9t8          <none>           <none>
   ```

7. Access the node via SSH as root and run `tcpdump`:

   ```
   tcpdump -A -v -i any host 10.111.1.122 and port 80
   ```

8. Create a Deployment client in the same namespace so that it is part of the mesh network:
   - If using a public registry image:

     ```
     d8 k -n test-istio-mtls create deployment client --image=docker.io/library/alpine:3.21 --
     /bin/sh -c "sleep infinity"
     ```

   - If using an all-in-one image (replace with your registry address):

     ```
     d8 k -n test-istio-mtls create deployment client --
     image=registry.company.network/localrepo/all-in-one-image:0.1 -- /bin/sh -c "sleep
     infinity"
     ```

9. After the Pod is created, make a request to the `webserver` service:

   ```
   d8 k -n test-istio-mtls exec -ti deployments/client -- wget -S --spider --timeout 1
   webserver`
   ```

   The `tcpdump` output will show only encrypted traffic:

```
10.111.91.68.http > 10.111.91.167.54204: Flags [P.], cksum 0xd7c7 (incorrect -> 0xe83a), seq
1:3033, ack 518, win 134, options [nop,nop,TS val 3160034944 ecr 2239691194], length 3032: HTTP
E....]@.@.T.
o[D
o[..P..'...;.i8..........
.ZN..~......z...v...T......d.4p
..6j..V..G...0 d..U h...e....W.`.....4....;.......T.......3.$...
[..%ex.....i<...*.[..C...jC....p.+..............Nt ..............
{..P%.+.,L......A........B..7}.v../...rg.o.)5......E.X;K8..%,..yb.^f...^+..Ble.
j..w3 S......}..
|3...+.=...;H..#...Y...3h.....:.9.w.A.4X..g...|S;.`<....%.y..4.....D.m...../6.7
[.......!+.J........2._r.D.C>`.3A..... ..H.^1...P1N....)..%...
r.=..s.FB..i^..^....4{;...ED_.\...g`l,q........U*0.#..(......).h..|sc...
.c(..kd....5C@..5.Z.L..C2
@.7>..y%....R.5........j9........!........H.a....b..].0..~..0x7..+N...o.6......
.. (%7:.Xr..N......'.O.a.j&v..Ba.....t..q.
=U.t.fU.].......2g....Eat.D.3n........*G.N..!LY.......n)......._jL..9RD..gT.lX..p&..=.d..Tq%....qF
`.....'..|..$!g..j.d. N%tb^...@- ..`...S:..D.....y...ckA.;}Y,.....X..6...=
[,..|AD..._}......W.. ..u3G...<. ..&.0j.,.'!.#
....w..bx.............}U.;..y....J.K..fQ.#]..3.V..=.d._.'....q.;!.9.N......n.7.Zi.>....@.
..].u.A}.;.....c..s......d.*=..G..9......Nt@....v..s.>.
...h....Cm.Z.......n......L.......-
_.......r...%Z.....h..........`..8A....yt.t..2d....oH?.1.O&.J....F..b.OV......
.......E1H..%~..2.H..{.I...=.I.*..2y1p0h..........P.....@r....vk.!".......
{..`.3..<,.r&L.....M...t...;.z...Q...1+.,........:._L.V.....e..
..!]\..6*}....vT.A>.....A0.....{.?}AH.+R..g.
[=?.X...|94....S+!..e...*..M`f...o..b.K#5.....@...5.......o.(.
...
```

10. Create a namespace without the `istio-injection=enabled` label and make a request to the `webserver.test-istio-mtls` service (a request is made from a Pod that is not part of the mesh network):

    ```
    d8 k create namespace test-istio-mtls-without-injection
    ```

11. Add a Deployment:

    - If using a public registry image:

      ```
      d8 k -n test-istio-mtls-without-injection create deployment alpine --
      image=docker.io/library/alpine:3.21 -- /bin/sh -c "sleep infinity"
      ```

    - If using an all-in-one image (replace with your registry address):

      ```
      d8 k -n test-istio-mtls create deployment client --
      image=registry.company.network/localrepo/all-in-one-image:0.1 -- /bin/sh -c "sleep
      infinity"
      ```

12. After the Pod is created, make a request to the `webserver.test-istio-mtls` service:

    ```
    d8 k -n test-istio-mtls-without-injection exec -ti deployments/alpine -- wget -S --spider --
    timeout 1 webserver.test-istio-mtls`.
    ```

    The `tcpdump` output will show only unencrypted (plain text) requests and responses:

```
10.111.91.211.54424 > 10.111.91.68.http: Flags [P.], cksum 0xcc77 (incorrect -> 0xee57), seq
1:93, ack 1, win 128, options [nop,nop,TS val 305286128 ecr 3834507600], length 92: HTTP, length:
92
GET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

E...y.@.?...
o[.
o[D...P....<Nb......w.....
.2K....PGET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

09:14:20.960302 lxc4f4a182c887c In  IP (tos 0x0, ttl 64, id 33396, offset 0, flags [DF], proto TCP
(6), length 52)
10.111.91.68.http > 10.111.91.211.54424: Flags [.], cksum 0xcc1b (incorrect -> 0xa850), ack 93,
win 128, options [nop,nop,TS val 3834507601 ecr 305286128], length 0
E..4.t@.@..Z
o[D
o[..P..<Nb.............
...Q.2K.
09:14:20.964496 lxc4f4a182c887c In  IP (tos 0x0, ttl 64, id 33397, offset 0, flags [DF], proto TCP
(6), length 1003)
10.111.91.68.http > 10.111.91.211.54424: Flags [P.], cksum 0xcfd2 (incorrect -> 0x44eb), seq
1:952, ack 93, win 128, options [nop,nop,TS val 3834507605 ecr 305286128], length 951: HTTP,
length: 951
HTTP/1.1 200 OK
server: istio-envoy
date: Fri, 24 Jan 2025 09:14:20 GMT
content-type: text/html
content-length: 615
last-modified: Wed, 14 Aug 2024 05:42:07 GMT
etag: "66bc43af-267"
accept-ranges: bytes
x-envoy-upstream-service-time: 0
connection: close
x-envoy-decorator-operation: webserver.test-istio-mtls.svc.cluster.local:80/*

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Additional resources

- nginx.org
- nginx.com
- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring cluster network stack visualization

To visualize network interactions in the cluster between pods, services, and external resources, as well as to monitor activity and identify network issues, use the `cilium-hubble` module.

Once you enable the module, the network stack visualization becomes available through the Hubble web interface.

Additional resources

- Module `cilium-hubble` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Switching CNI from Flannel or Simple bridge to Cilium

> This guide is applicable to Deckhouse Platform version 1.75 and earlier.
>
> For DP version 1.76 and later, use the Switching CNI in the cluster guide.

> ⚠ The migration requires a **full cluster downtime**: all pods and nodes will be restarted. Estimated time to complete is **about 1 hour**.

To perform the migration, follow these steps:

1. Enable the `cni-cilium` module:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cni-cilium
   spec:
     version: 1
     settings:
       tunnelMode: VXLAN
     enabled: true
   ```

2. Wait for the Cilium components to start:

   ```
   d8 k -n d8-cni-cilium get pods
   ```

   Example output:

```
NAME                          READY STATUS  RESTARTS    AGE
agent-5zzfv                   2/2   Running 5 (23m ago) 26m
agent-gqb2b                   2/2   Running 5 (23m ago) 26m
agent-wtv4p                   2/2   Running 5 (23m ago) 26m
operator-856d69fd49-mlglv 2/2   Running 0           26m
safe-agent-updater-26qpk  3/3   Running 0           26m
safe-agent-updater-qlbrh  3/3   Running 0           26m
safe-agent-updater-wjjr5  3/3   Running 0           26m
```

If the Cilium agent pods do not transition to the `Ready` state:

- Save the `d8-kube-proxy` manifest:

  ```
  d8 k -n kube-system get ds d8-kube-proxy -oyaml > d8-kube-proxy.yaml
  ```

- Run the recovery actions:

  ```
  d8 k delete validatingadmissionpolicies.admissionregistration.k8s.io label-
  objects.deckhouse.io
  d8 k -n kube-system delete ds d8-kube-proxy
  d8 k -n d8-cni-cilium delete po -l app=agent
  ```

- If Cilium startup still takes more than 5 minutes, restart `kube-proxy`:

  ```
  d8 k -n kube-system delete pod -l k8s-app=kube-proxy
  ```

3. Save the DaemonSet manifest:

   ```
   d8 k -n d8-cni-simple-bridge get ds simple-bridge -o yaml > simple-bridge.yaml
   # or
   d8 k -n d8-cni-flannel get ds flannel -o yaml > flannel.yaml
   ```

4. Disable `cni-simple-bridge` or `cni-flannel`. Perform this step only after the Cilium pods have reached the `Ready` state:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-
   simple-bridge
   # or
   d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-
   flannel
   ```

5. Delete the `validating webhook` (if present):

   ```
   d8 k delete validatingwebhookconfigurations.admissionregistration.k8s.io d8-deckhouse-validating-
   webhook-handler-hooks
   ```

6. Delete the namespace of the old CNI (if it was not removed automatically):

   ```
   d8 k delete ns d8-cni-simple-bridge
   # or
   d8 k delete ns d8-cni-flannel
   ```

7. (Optional) Clean up artifacts of the old CNI. If required, remove the interface and flush iptables rules:

```
ip link delete cni0

iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -t raw -F
iptables -t raw -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
```

8. Restart all pods:

```
d8 k delete pods -A --all --wait=false
```

9. Configure internal LoadBalancers (if used). For all TargetGroups that belong to the internal load balancer, set:

```
Preserve client IP addresses = Off
```

10. Reboot the cluster nodes one by one, including the master nodes.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Incoming traffic balancing

This section describes the approaches to balancing incoming traffic in Deckhouse Platform (DP):

- NLB (Network Load Balancer) — operates at the network level, routing traffic based on IP addresses and ports without inspecting request contents.
- ALB (Application Load Balancer) — operates at the application level, analyzing HTTP(S) headers, paths, and domains. It supports SSL termination and content-based routing.

Network-level load balancing (NLB)

NLB-based load balancing can be implemented in two ways:

- Using an external load balancer provided by a cloud provider.
- Using the built-in `metallb` balancer, which works in both cloud and bare-metal clusters.

Application-level load balancing (ALB)

For application-level traffic balancing, DP provides following solutions:

- Ingress NGINX Controller (via the `ingress-nginx` module).
- Kubernetes Gateway API (`alb` module).
- Istio (via the `istio` module).

Difference between the Kubernetes Gateway API and an API gateway

Kubernetes Gateway API and an API gateway serve different purposes:

- The Kubernetes Gateway API is a set of Kubernetes resources (a specification) that describe how inbound traffic is routed to services. It is a configuration interface implemented by controllers, and the successor to the Ingress API.
- An API gateway is an architectural component (or product) that aggregates several application APIs behind a single entry point and centralizes cross-cutting functions such as authentication,

authorization, and request rate limiting for API consumers.

In other words, the Kubernetes Gateway API describes how to configure traffic routing, while an API gateway is a type of infrastructure that handles that traffic. Some API gateways can be configured through the Kubernetes Gateway API. The `alb` module is an implementation of the Kubernetes Gateway API.

Role separation in the Gateway API model

When using the `alb` module, responsibilities are typically split as follows:

- Cluster administrator — deploys cluster-scoped gateway infrastructure with ClusterALBInstance.
- Namespace administrator — deploys namespaced gateway infrastructure with ALBInstance and configures how traffic is accepted with ListenerSet (hostname, TLS, ports).
- Application developers — configure routing to applications with HTTPRoute and other route objects.

In a typical cluster-wide gateway scenario, the namespace administrator creates the ListenerSet, and application developers create the HTTPRoute. The same person may perform both roles if they have the required permissions.

Choosing an ALB implementation

The table below lists the criteria for choosing an ALB implementation.

Criterion	Ingress NGINX	Gateway API	Istio
Publishing API	Ingress + annotations	Gateway, ListenerSet, routes	Gateway, VirtualService, DestinationRule
Protocols	HTTP/HTTPS, gRPC (Ingress)	HTTP/HTTPS, gRPC, TLS, TCP, UDP	HTTP/HTTPS, gRPC, TCP (Istio Gateway)
Role separation	IngressClass + Ingress	ClusterALBInstance/ALB Instance → ListenerSet → routes	IngressIstioController + Istio Gateway
Service mesh	No	Optional (sidecar on the gateway proxy)	Yes
Typical use case	Classic Ingress, minimal change	New apps, multitenancy, gradual migration from Ingress	Canary, mTLS, tracing, advanced routing

Comparison of the ingress-nginx and alb modules

Both modules solve the same task — receiving and routing external traffic to applications — but rely on different standards: `ingress-nginx` uses the Ingress API with annotations, while `alb` uses the Kubernetes Gateway API. The modules can be used in a cluster simultaneously. The table below compares their capabilities in the current versions.

Service domains (web interfaces of DP components and modules via `publicDomainTemplate`) and application domains (routes owned by application developers) are configured differently. Details are in Publishing service domains.

Capability	**`ingress-nginx`**	**`alb`**
Routing standard	Ingress API with annotations	Kubernetes Gateway API
Proxy implementation	nginx	Envoy Proxy
Lifecycle stage	General Availability	Preview
Development	Maintenance mode: the upstream Ingress NGINX project no longer develops new features, while DP provides security updates	Actively developed
Minimum DP version	Available in all supported versions	1.76
DP editions	All editions	All editions

Capability	ingress-nginx	alb
Role separation model	Cluster administrator, namespace administrator	Cluster administrator, namespace administrator, application developers
Multiple independent entry points	Multiple Ingress controllers selected via `ingressClass`	Multiple Gateway objects selected via `gatewayName`; cluster-scoped and namespaced gateways
HTTP/HTTPS (HTTP/1.1, HTTP/2, HTTP/3)	Yes	Yes (enabling HTTP/3)
WebSocket	Yes	Yes
gRPC	Yes	Yes
FastCGI	Yes	No
TCP	No	Yes (TCPRoute)
UDP	No	Yes (UDPRoute)
TLS passthrough	Yes	Yes (TLSRoute)
Proxy Protocol	Yes	Yes
Traffic ingress methods	`LoadBalancer`, `HostPort`, and `HostWithFailover` inlets	`LoadBalancer` and `HostPort` inlets
Automatic TLS certificate issuance (cert-manager)	Yes	Yes
HTTPS policy tuning (TLS versions, ciphers, HSTS)	Yes	TLSv1.2/1.3 by default; HSTS via a response-header annotation
WAF	ModSecurity at the controller or Ingress level	ModSecurity/Coraza at the route level, OWASP CRS preset
External authentication	Yes	Yes
IP allowlist	Yes	Yes
Basic authentication	Yes	Yes
Request rate limiting	Yes	Yes
Session affinity	Yes	Yes
GeoIP	Geo-based request statistics in metrics	Adding GeoIP fields to headers based on MaxMind databases
Prometheus metrics and Grafana dashboards	Yes, detailed by namespace, vhost, Ingress resource, and location	Yes: Envoy Proxy metrics and dashboards for requests, routes, and upstreams
OpenTelemetry tracing	Yes	Yes (configuration)

Next steps

To publish an application, follow these steps:

1. Choose an ALB implementation based on the criteria in the table above: Ingress NGINX, Gateway API, or Istio.
   - Istio — when you need service-mesh traffic management (canary routing, mTLS between Pods).
   - Gateway API — when you need a model with role separation and protocols beyond classic Ingress.
   - Ingress NGINX — when you need a mature Ingress-based ALB.
2. Enable and configure the corresponding module. For Gateway API, complete the “Steps to take before enabling and configuring ALB in a cluster”.
3. Publish applications using the ALB user guides (Gateway API, Ingress NGINX, or Istio).
4. To move from Ingress NGINX to Gateway API, follow Migrating from ingress-nginx to alb.

Additional resources

- ALB with Ingress NGINX Controller
- ALB with Kubernetes Gateway API
- Migrating from ingress-nginx to alb
- ALB with Istio
- Utilizing Application Load Balancer (ALB)
- Module `alb` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation
- Module `metallb` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Balancing with MetalLB

The `metallb` module implements support for LoadBalancer-type services in Deckhouse Platform (DP) clusters. It is suitable for both bare-metal clusters and cloud environments where built-in load balancers by providers are unavailable.

Two operating modes are supported:

- **Layer 2**: An enhanced version of the standard L2 mode in MetalLB, allowing multiple public addresses to be used for services.
- **BGP**: Fully based on the MetalLB solution and available only in DP Enterprise Edition.

Layer 2 mode

How it works

In Layer 2 mode, one or more cluster nodes receive traffic for a service from the public network. From the network layer's perspective, each of these nodes has multiple IP addresses assigned to its network interface. This is implemented by the module responding to ARP requests (for IPv4) and NDP requests (for IPv6).

The main advantage of this mode is universality. It works in any Ethernet network without requiring specialized hardware.

Advantages over classic MetalLB

In classic MetalLB (L2 mode), when creating a LoadBalancer service, load balancing is achieved by having a single node in the cluster reply to ARP requests for the public IP. This means:

- Only one node handles all incoming traffic for the given IP at a time.
- The leader node becomes a bottleneck with no horizontal scaling.
- If the leader node fails, all active connections are dropped during the switchover to the new node.

The `metallb` module removes these limitations. It provides the MetalLoadBalancerClass resource, which lets you:

- Bind a group of nodes to an IP address pool using `nodeSelector`.
- Create a standard LoadBalancer Service and specify the name of the target MetalLoadBalancerClass.
- Define the number of IP addresses for L2 announcement via an annotation.

This approach means:

- The application receives multiple public IPs, which must be added as A records in DNS.
- To scale, add more load balancer nodes. Associated Services will be created automatically — only adding the IPs to the application domain's A records is required.
- If one balancer fails, only part of the traffic is rerouted, without a complete connection drop.

Behavior comparison

Feature	Classic MetalLB (L2)	New module with MetalLoadBalancerClass
Traffic handling	Single node (leader)	Multiple nodes
Scalability	No	Yes
Fault tolerance	All connections dropped	Part of the traffic is rerouted smoothly
Number of public IPs	One	Multiple (configurable)
DNS configuration	One A record	Multiple A records

Example of using MetalLB in L2 LoadBalancer mode

1. Enable the `metallb` module:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: metallb
   spec:
     enabled: true
     version: 2
   ```

2. Deploy the application to expose:

   ```
   d8 k create deploy nginx --image=nginx
   ```

3. Create a MetalLoadBalancerClass resource:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: MetalLoadBalancerClass
   metadata:
     name: ingress
   spec:
     addressPool:
       - 192.168.2.100-192.168.2.150
     isDefault: false
     l2:
       # Network interfaces that announce the assigned IP addresses.
       interfaces:
         - eth1
     nodeSelector:
       # Selects nodes that run the L2 load balancer.
       node-role.kubernetes.io/loadbalancer: ""
     type: L2
   ```

   In this example, `spec.nodeSelector` selects nodes with the `node-role.kubernetes.io/loadbalancer` label. Make sure the nodes intended to announce the Service IP addresses have this label.

   ⚠ If the `spec.l2.interfaces` parameter is not set or contains an empty list, the assigned IP addresses are announced from all network interfaces of the selected nodes. Specify the interfaces to announce the addresses from.

   If the selected nodes have taints with custom keys, allow the `speaker` to run on them as described in the Running speakers on nodes with custom taints section.

4. Create a Service resource with the annotation and name MetalLoadBalancerClass:

> **i** When creating a Service resource, you can omit the annotations `network.deckhouse.io/l2-load-balancer-external-ips-count` and `network.deckhouse.io/load-balancer-ips` from the examples below. In this case, the service will automatically be assigned 1 random address from the pool.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  # Selects the MetalLoadBalancerClass named ingress.
  # Can be omitted if the cluster has a MetalLoadBalancerClass with isDefault:
true
  # and BGP LoadBalancer mode is not used.
  loadBalancerClass: ingress
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

If you need to assign specific IP addresses from the pool to the service...

> **i** If the service needs to allocate specific IP addresses from the pool, use the annotation `network.deckhouse.io/load-balancer-ips`. If there is more than one desired address, there must also be an annotation `network.deckhouse.io/l2-load-balancer-external-ips-count`, which must specify the number of addresses allocated from the pool (it must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`).

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # The number of addresses that will be allocated from the pool declared in
MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # A list of addresses from the pool declared in MetalLoadBalancerClass that will
be allocated to the service.
    network.deckhouse.io/load-balancer-ips:
"192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  # Selects the MetalLoadBalancerClass named ingress.
  # Can be omitted if the cluster has a MetalLoadBalancerClass with isDefault: true
  # and BGP LoadBalancer mode is not used.
  loadBalancerClass: ingress
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

As a result, the LoadBalancer Service will be assigned the configured number of addresses:

```
d8 k get svc
```

Example output:

```
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)          AGE
nginx-deployment        LoadBalancer   10.222.130.11
192.168.2.100,192.168.2.101,192.168.2.102   8000:30544/TCP   11s
```

The obtained `EXTERNAL-IP` can be set as A records for the application domain:

```
curl -s -o /dev/null -w "%{http_code}" 192.168.2.100:8000
curl -s -o /dev/null -w "%{http_code}" 192.168.2.101:8000
curl -s -o /dev/null -w "%{http_code}" 192.168.2.102:8000
```

Example output:

```
200
```

BGP mode

In BGP mode, `metallb` provides LoadBalancer-type services in Kubernetes clusters deployed on physical infrastructure. Service IP addresses are announced directly to routers (or top-of-rack switches) via the BGP protocol.

How it works

Configuration

- Define an IP address pool available for Service allocation.
- Specify BGP session parameters: the Kubernetes cluster Autonomous System (AS) number, router (peer) IP addresses, peer AS numbers, and passwords for authentication (if needed).
- For each IP pool, define specific announcement parameters, such as community strings.

Establishing BGP sessions

- On each Kubernetes node running `metallb`, the `speaker` component establishes BGP sessions with the configured routers.
- Route information is exchanged between `metallb` and the routers.

Assigning IP addresses to Services

- When a LoadBalancer Service is created, `metallb` selects a free IP address from the configured pool and assigns it to the Service.
- The `controller` component tracks Service changes and manages IP address assignments.

Announcing IP addresses

- After assigning an IP address, the `speaker` component on one of the nodes (leader for the Service) announces it via the established BGP sessions.
- Routers receive the announcement and update their routing tables to direct traffic to the corresponding node.

Traffic distribution

- Routers use Equal-Cost Multi-Path (ECMP) protocols or other load balancing algorithms to distribute traffic between nodes announcing the same Service IP.
- Once delivered to the node, incoming traffic is forwarded to the Service pods using the CNI mechanisms (iptables/IPVS, eBPF, etc.).

Benefits of using BGP

- BGP is supported by most network equipment.
- The network can include multiple routers and many nodes.
- Traffic balancing via ECMP.

- If a node stops announcing an IP, routers automatically redirect traffic to other nodes with the same IP.

Drawbacks of using BGP

- More complex to configure than ARP/GARP announcements (Layer 2 mode).
- Routers must support BGP and ECMP.

Example of using MetalLB in BGP LoadBalancer mode

1. Enable the `metallb` module and configure the required parameters:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: metallb
   spec:
     enabled: true
     version: 3
   ---
   apiVersion: network.deckhouse.io/v1alpha1
   kind: MetalLoadBalancerPool
   metadata:
     name: mypool
   spec:
     addresses:
       - 192.168.219.100-192.168.219.200
   ---
   apiVersion: network.deckhouse.io/v1alpha1
   kind: MetalLoadBalancerBGPPeer
   metadata:
     name: myrouter
   spec:
     peerAddress: 172.18.18.10
     peerASN: 64601
     myASN: 64600
     holdTime: 3s
   ---
   apiVersion: network.deckhouse.io/v1alpha1
   kind: MetalLoadBalancerConfiguration
   metadata:
     name: defaults
   spec:
     mode: BGP
     nodeSelector:
       node-role.deckhouse.io/metallb: ""
     bgp:
       peerNames:
         - myrouter
     advertisements:
       - poolNames:
           - mypool
   ```

   In this example, `spec.nodeSelector` selects nodes with the `node-role.deckhouse.io/metallb` label. Make sure the nodes intended to run the `speaker` pods have this label.

   If the selected nodes have taints with custom keys, allow the `speaker` to run on them as described in the Running speakers on nodes with custom taints section.

2. Configure BGP peering on the network equipment.

Running speakers on nodes with custom taints

The `speaker` pods announce Service IP addresses and run as a DaemonSet on the nodes selected by

the `nodeSelector` parameter: of MetalLoadBalancerClass in Layer 2 mode and of MetalLoadBalancerConfiguration in BGP mode. The `speaker` pods tolerate only the following taints known to DP:

- `node-role.kubernetes.io/master`
- `node-role.kubernetes.io/control-plane`
- `node.deckhouse.io/etcd-arbiter`
- `dedicated.deckhouse.io` and `dedicated` with any value
- `DeletionCandidateOfClusterAutoscaler` and `ToBeDeletedByClusterAutoscaler`

If a selected node has a taint with any other key, the `speaker` is not scheduled on it, and the IP addresses are not announced from that node.

To run the `speaker` on such nodes, follow these steps:

1. Add the taint key to the `customTolerationKeys` parameter of the `global` ModuleConfig. The key must match the taint key set on the nodes, for example, in the `spec.nodeTemplate.taints` parameter of NodeGroup. The parameter also allows the other system components (CNI, CSI, monitoring) to run on such nodes.

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: global
   spec:
     version: 3
     settings:
       modules:
         placement:
           customTolerationKeys:
             - dedicated.example.com
   ```

2. Check that the `speaker` pods are running on the tainted nodes.

   For Layer 2 mode:

   ```
   d8 k -n d8-metallb get pods -l app=l2lb-speaker -o wide
   ```

   For BGP mode:

   ```
   d8 k -n d8-metallb get pods -l app=speaker -o wide
   ```

Additional resources

- Global parameters
- Module `metallb` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Balancing in clusters on cloud platforms

A list of supported providers:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- OpenStack
- Huawei Cloud
- VMware Cloud DirectorExperimental
- VMware vSphere

- Yandex Cloud
- zVirtExperimental

Configuring incoming traffic balancing in clusters on cloud platforms involves creating an Ingress controller with specified LoadBalancer parameters. Based on these settings, the cloud provider automatically creates an external load balancer. In the cluster, a Service resource is created through which the external load balancer routes traffic to applications.

Example of creating an Ingress controller for the OpenStack provider

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Example of creating a ClusterIP-type Service

```
apiVersion: v1
kind: Service
metadata:
  name: backend-resolver-cluster-ip
spec:
  ports:
  - name: http
    port: 8000
    protocol: TCP
  selector:
    app: lab-4-backend
  type: ClusterIP
```

Example for VK Cloud

The following example is relevant when the internal balancer would be used.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

ALB with Ingress NGINX Controller

The `ingress-nginx` module is used to implement ALB using the Ingress NGINX Controller.

> In 2025, Ingress NGINX was placed in maintenance mode, with no plans for active development of new features. Further evolution of inbound traffic load balancing in Kubernetes is focused on the Gateway API.
>
> This does not apply to the module as part of Deckhouse Platform: the module is maintained by the Deckhouse Platform team, including security updates. Details are in “Module support and security”.
>
> Step-by-step migration to Gateway API is in Migrating from ingress-nginx to alb.

The `ingress-nginx` module installs the Ingress NGINX Controller and manages it with custom resources. If there is more than one node available for hosting the Ingress controller, it is deployed in the HA mode, taking into account the infrastructure specifics of both cloud and bare-metal environments, as well as various Kubernetes cluster types.

The module supports running and configuring several Ingress NGINX controllers simultaneously (one of the controllers is the primary one. You can create as many additional controllers as you want). This approach allows you to separate extranet and intranet Ingress resources of applications.

Traffic termination options

Traffic to `ingress-nginx` can be routed in several ways:

- Directly without the use of an external load balancer.
- Using an external LoadBalancer. The following variants are supported:
  - Qrator
  - Cloudflare
  - AWS LB
  - GCE LB
  - ACS LB
  - Yandex LB
  - OpenStack LB

HTTPS termination

The module allows you to manage HTTPS security policies for each of the Ingress NGINX controllers, including:

- HSTS parameters
- Available SSL/TLS versions and encryption protocols

The module is integrated with the `cert-manager` module. Thus, it can get SSL certificates automatically and pass them to Ingress NGINX controllers for further use.

Monitoring and statistics

The current `ingress-nginx` implementation has a Prometheus-based system for collecting statistical data with the following set of metrics:

- Total response time and backend response time separately
- Response codes
- Number of repeated requests (retries)
- Request and response sizes
- Request methods
- `content-types`
- Geography of requests, etc.

The data can be grouped by the:

- `namespace`
- `vhost`
- `ingress` resources
- `location` (in nginx)

All graphs are grouped by Grafana dashboards. From any graph you can open a more detailed view: for example, from a `namespace` statistics view, you can click through to the corresponding `vhost` dashboard for more detail, and continue down the hierarchy.

Statistics

Basic principles of collecting statistics

1. At the `log_by_lua_block` stage, the module calculates the necessary metrics for each request and stores them in a buffer (each NGINX worker has its own buffer).
2. At the `init_by_lua_block` stage, each NGINX worker starts a process that sends data in `protobuf` format via TCP socket to the `protobuf_exporter` every second (developed by Deckhouse Platform).
3. `protobuf_exporter` runs as a sidecar container in the Ingress controller pod, receives `protobuf` messages, parses and aggregates them, and exports metrics for Prometheus.
4. Prometheus scrapes metrics every 30 seconds from both the Ingress controller and the `protobuf_exporter`. This scraped data is what statistics is based on.

Metrics structure and representation

All collected metrics include service labels identifying the controller instance: `controller`, `app`, `instance`, and `endpoint` (visible in `/prometheus/targets`).

- All non-geo metrics exported by `protobuf_exporter` are provided at three detail levels:
  - `ingress_nginx_overall_*`: Top-level aggregated metrics (non-detailed, all metrics have the following labels: `namespace`, `vhost`, `content_kind`).
  - `ingress_nginx_detail_*`: In addition to overall metrics, adds `ingress`, `service`, `service_port`, and `location`.
  - `ingress_nginx_detail_backend_*`: Backend-level metrics. In addition to detail metrics, adds the `pod_ip` label.
- Metrics collected for overall and detail levels:
  - `*_requests_total`: Total requests (extra labels: `scheme`, `method`).
  - `*_responses_total`: Number of responses (extra label: `status`).
  - `*_request_seconds_{sum,count,bucket}`: Response time histogram.
  - `*_bytes_received_{sum,count,bucket}`: Request size histogram.
  - `*_bytes_sent_{sum,count,bucket}`: Response size histogram.
  - `*_upstream_response_seconds_{sum,count,bucket}`: Upstream service response time histogram (total for multiple upstreams).
  - `*_lowres_upstream_response_seconds_{sum,count,bucket}`: Simplified histogram (for visualization; can't be used for quantiles).
  - `*_upstream_retries_{count,sum}`: Number and total of backend retries.
- Metrics collected for overall level:
  - `*_geohash_total`: Request counts per geohash (additional labels: `geohash`, `place`).
- Metrics collected for detail backend level:
  - `*_lowres_upstream_response_seconds`: Simplified response time histogram for overall and detail.
  - `*_responses_total`: Number of responses (additional label: `status_class`, not just `status`).
  - `*_upstream_bytes_received_sum`: Total size of data received from backends.

Load balancing configuration examples

Use the IngressNginxController custom resource to configure load balancing.

- AWS (NLB)
- GCP, Yandex Cloud, and Azure
- OpenStack
- VK Cloud
- Bare metal (HostWithFailover)
- Bare metal with external LB
- MetalLB BGP
- MetalLB L2

Example for AWS (Network Load Balancer)

When setting up the balancer, all available zones in the cluster are used.

Each zone’s balancer receives its own public IP. If a zone has an Ingress controller instance, its IP is added to the load balancer’s domain name as an A record.

If no instances remain in a zone, that IP is removed from DNS.

If only one Ingress controller instance exists in a zone, its IP is temporarily removed from DNS during pod restarts.

Example IngressNginxController with the `LoadBalancer` inlet and AWS NLB annotations:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

Example for GCP, Yandex Cloud, and Azure

IngressNginxController with the `LoadBalancer` inlet:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
```

i In GCP, nodes must have an annotation allowing external connections for NodePort services.

Example for OpenStack

IngressNginxController with the `LoadBalancerWithProxyProtocol` inlet and OpenStack Proxy Protocol annotations:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main-lbwpp
spec:
  inlet: LoadBalancerWithProxyProtocol
  ingressClass: nginx
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
```

Example for VK Cloud

Use this configuration for an internal cloud balancer (without a public address).

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker
```

Example for bare metal

IngressNginxController with the `HostWithFailover` inlet on frontend nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostWithFailover
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

Example for bare metal with external load balancer

Use this configuration with Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp, or other external load balancers.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

Example for bare metal (MetalLB in BGP LoadBalancer mode)

IngressNginxController with the `LoadBalancer` inlet for use with MetalLB in BGP mode:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

When using MetalLB, its speaker pods (MetalLB components that announce IP addresses) must run on the same nodes as the Ingress controller pods.

To preserve the real client IP addresses, the Ingress controller Service should be created with `externalTrafficPolicy: Local` to avoid inter-node SNAT. In this configuration, MetalLB speaker will only announce the Service from nodes running target pods.

Example ModuleConfig for the `metallb` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
  settings:
    speaker:
      nodeSelector:
        node-role.deckhouse.io/frontend: ""
      tolerations:
        - effect: NoExecute
          key: dedicated.deckhouse.io
          value: frontend
```

Example for bare metal (MetalLB in L2 LoadBalancer mode)

1. Enable the `metallb` module:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: metallb
   spec:
     enabled: true
     version: 2
   ```

2. Create a MetalLoadBalancerClass resource:

   > **i** MetalLB balancers should be placed on the same nodes as Ingress controllers. In a cluster with role separation, frontend nodes are used for this purpose. To deploy Ingress controllers and MetalLB load balancers on frontend nodes, set the label `node-role.deckhouse.io/frontend: ""` in `nodeSelector`.

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  l2:
    # Network interfaces that announce the assigned IP addresses.
    interfaces:
      - eth1
  nodeSelector:
    # Selects nodes that run the L2 load balancer.
    node-role.deckhouse.io/frontend: ""
  type: L2
```

3. Create an IngressNginxController resource:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    loadBalancerClass: ingress
    annotations:
      # Number of addresses to allocate from the pool defined in
MetalLoadBalancerClass.
      network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
      operator: Equal
```

> **i**
>
> When creating an ingress controller, you can also specify certain IP addresses from the pool that will be assigned to its Service. Use the annotation `network.deckhouse.io/load-balancer-ips`.
>
> If you need more than one address, also set `network.deckhouse.io/l2-load-balancer-external-ips-count` to the number of addresses allocated from the pool. That value must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`.
>
> A configuration example is provided in the Creating a Service and assigning it specific IP addresses from the pool section.

Deckhouse Platform will create a LoadBalancer Service with the specified number of IPs:

```
d8 k -n d8-ingress-nginx get svc
```

Example output:

```
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                       AGE
main-load-balancer      LoadBalancer   10.222.130.11
192.168.2.100,192.168.2.101,192.168.2.102   80:30689/TCP,443:30668/TCP   11s
```

Example of segregating access between public and administrative zones

In many applications, the same backend serves both the public part and the administrative interface. For example:

- `https://example.com` is the public zone;
- `https://admin.example.com` is the administrative zone, access to which must be restricted (`ACL`, `mTLS`, `IP whitelist`, and so on).

For this scenario, offload administrative traffic to a separate Ingress controller (with a dedicated Ingress class if necessary) and restrict access to it by using the `spec.acceptRequestsFrom` parameter.

- Single Ingress controller
- Separate Ingress controllers

Specifics of using a single Ingress controller

The example below shows a single Ingress controller serving requests from both the public zone and the administrative interface.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: nginx # The Ingress resource for administrative traffic is
associated with the same Ingress controller as the Ingress resource for public
traffic.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: nginx # The Ingress resource for public traffic is associated
with the same Ingress controller as the Ingress resource for administrative traffic.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

With processing and forwarding of X-Forwarded-* headers enabled, the backend can rely on the `x-forwarded-host` header when making authorization decisions. In the example above, public Ingress traffic can reach the administrative zone via `x-forwarded-host`. Therefore, requests to the Ingress controller must come only from trusted sources.

Using separate Ingress controllers

To avoid that situation, do the following:

- Configure access rules at the Ingress resource level.
- Use separate Ingress controllers.
- Restrict which source addresses are allowed to connect to the Ingress controllers.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: admin-nginx # The Ingress resource for administrative traffic is
associated with a separate Ingress controller.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: public-nginx # The Ingress resource for public traffic is
associated with a separate Ingress controller.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

Example of an Ingress controller that serves administrative Ingress resources and accepts connections only from specified subnets:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: admin
spec:
  ingressClass: admin-nginx
  inlet: HostPort
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

In this example:

- The Ingress controller is exposed on node ports through the `HostPort` inlet.
- The `acceptRequestsFrom` parameter allows connections to the controller only from the listed

subnets.
- Even if an external load balancer or client can set its own `X-Forwarded-*` header values, the decision whether to allow the connection to reach the controller is made based on the actual source address, not on headers.
- Administrative Ingress resources (in this example `admin-ingress`) are served by this controller according to the configured Ingress class.

Example of an Ingress controller that serves Ingress resources for public traffic:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: public
spec:
  ingressClass: public-nginx
  inlet: HostPort
  hostPort:
    httpPort: 8080
    httpsPort: 8443
    behindL7Proxy: true
```

Module support and security

The `ingress-nginx` module is covered by Deckhouse Platform maintenance for the entire platform support lifecycle, regardless of the upstream project's development status. The Deckhouse Platform team tracks CVEs in the controller and its dependencies — NGINX, Lua modules, and base images — and delivers fixes in platform releases.

For compliance with PCI DSS expectations regarding vendor support and vulnerability remediation timelines, Flant is the responsible vendor of the module. Deckhouse Platform certification with FSTEC of Russia also covers vulnerability management processes and the release of security updates.

Additional resources

- Migrating from ingress-nginx to alb
- ALB with Kubernetes Gateway API
- Utilizing Application Load Balancer (ALB)
- Module `ingress-nginx` documentation
- Module `ingress-nginx` custom resources
- Module `ingress-nginx` documentation
- Module `metallb` documentation
- Module `cert-manager` documentation
- Module `metallb` parameters
- Module `metallb` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

ALB with Kubernetes Gateway API

To implement an Application Load Balancer (ALB) using the Kubernetes Gateway API, the `alb` module is used.

The `alb` module implements an Application Load Balancer and allows you to publish applications through Kubernetes Gateway API. It deploys and configures the infrastructure for receiving and routing external requests, and also verifies the user configuration of the Gateway API.

> **i** ALBs created using the Kubernetes Gateway API can be used in a cluster alongside ALBs created using the Ingress NGINX Controller. Details are in "Using with other modules and third-party solutions".

Overview and scheme

The module is built on the Kubernetes Gateway API — an API for inbound traffic routing that extends the Ingress API model. A ClusterALBInstance or ALBInstance creates a managed Gateway. A ListenerSet bound to that Gateway describes handlers for incoming requests. Routes direct traffic to application services.

The module supports:

- A single declarative API for HTTP/HTTPS, gRPC, TCP, UDP, and TLS passthrough.
- A separation of responsibilities between the cluster administrator (ClusterALBInstance or ALBInstance), the namespace administrator (ALBInstance and ListenerSet — hostname, TLS, ports), and application developers (routes).
- Request-handling features: per-route WAF, external authentication, IP allowlists, rate limiting, session affinity, GeoIP, BackendTLSPolicy, Proxy Protocol, and HTTP/3.

Kubernetes Gateway API and an API gateway serve different purposes. The Kubernetes Gateway API is a set of Kubernetes resources that describe how inbound traffic is routed to services. An API gateway is an architectural component that aggregates application APIs behind a single entry point. The `alb` module is an implementation of the Kubernetes Gateway API.

For a capability comparison with `ingress-nginx`, open the “Comparison of the ingress-nginx and alb modules” section.

Object roles

Gateway API separates responsibilities between cluster and namespace administrators and application developers:

- Cluster administrator — manages traffic infrastructure through ClusterALBInstance or ALBInstance (cluster-wide Gateway).
- Namespace administrator — manages ALBInstance and ListenerSet (hostname, TLS, ports) within a namespace.
- Application developers — define routes (HTTPRoute, GRPCRoute, TLSRoute, TCPRoute, UDPRoute).

Why ListenerSet

ListenerSet is a Gateway API extension. The ListenerSet object describes system and user traffic handlers: hostname, TLS mode, port, and protocol. Each ListenerSet is linked to a parent Gateway through `spec.parentRef`. Routes are then attached to it.

Each Gateway object creates two default listeners: `d8-http` (port `80`) and `d8-https` (port `443`). They are intended for service tasks such as gateway availability checks or cert-manager HTTP-01 challenges. They are not recommended for publishing applications. Use ListenerSet for that purpose instead.

ClusterALBInstance and ALBInstance

When creating a Gateway managed object for publishing user applications, the custom resources ClusterALBInstance (a cluster-scoped object) and ALBInstance (a namespaced object) are used.

The characteristics and differences between them are described in the table:

	ClusterALBInstance	**ALBInstance**
Purpose	Deploy a cluster-wide Gateway object	Deploy a local Gateway object

	ClusterALBInstance	**ALBInstance**
Typical use case	- Common entry point (cluster-wide gateway). - System gateway for publishing web interfaces of Deckhouse Platform (DP) service components and other modules (may require “Steps before enabling”). - Platform gateway	Dedicated gateway for an application or team in a dedicated namespace
Supported inlet types	`LoadBalancer`, `HostPort`	`LoadBalancer`, `ClusterIP`
Proxy implementation	Envoy Proxy	Envoy Proxy
Deployment type	DaemonSet	Deployment
Placement of ListenerSet objects and routes	In any user namespace	In the same namespace as the ALBInstance object (required)
Access level	Cluster administrator	Namespace administrator

Creating a ClusterALBInstance object or an ALBInstance object results in creation of a managed Gateway object in the cluster. At the same time:

- Each Gateway object is served by at least one Envoy Proxy instance.
- Traffic reaches it through a Service of type `LoadBalancer` or directly by using `HostPort` parameters.
- Several ClusterALBInstance or ALBInstance objects may point to the same Gateway object through the `gatewayName` field. In that case, they describe one shared gateway. The request handling infrastructure may still differ depending on settings. You can think of `gatewayName` as an analog of `ingressClass` for IngressNginxController objects. The resulting configuration comes from the instance with the earliest `creationTimestamp` (that is, the one created first). The others get `conflictPorts: true` in status, with `conflictPortsOwner` naming the controlling instance.

Validating configuration

In addition to Gateway API infrastructure configuration, user settings are validated before being applied to prevent conflicts. This allows detecting conflicts between identical traffic handlers in different ListenerSet objects when they point to the same Gateway object.

Inlets

An inlet defines how external traffic reaches the Envoy Proxy that serves the Gateway object:

- `LoadBalancer` — traffic is accepted through a Service of type `LoadBalancer` (cloud providers or bare metal with MetalLB). Available for both ClusterALBInstance and ALBInstance.
- `HostPort` — traffic is accepted on node ports without an external load balancer. Available for ClusterALBInstance only.
- `ClusterIP` — traffic is accepted through a Service of type `ClusterIP`, without an external entry point (for example, for internal traffic). Available for ALBInstance only.

The inlet mapping table for migration from `ingress-nginx` is in “Inlet configuration”. More examples are in “Examples for different environments”.

How to configure ALB

Publishing an application includes enabling the module, creating a managed Gateway, a ListenerSet, and routes.

Steps before enabling

Before enabling and configuring ALB in a DP cluster, do the following:

- Verify that the requirements for the `alb` module are met.
- If you need to publish service domains — web interfaces of DP service components and other modules — set the global parameter `publicDomainTemplate`. Without this parameter, system HTTPRoute, Gateway, and ListenerSet objects for service domains will not work correctly, and the web interfaces will not be published. If you do not need to publish service domains, you can leave this parameter unset. Details are in “Publishing service domains”.
- Check API version compatibility in “Alongside third-party Gateway API implementations” if such solutions are already used in the cluster.

- On bare metal, for the `LoadBalancer` inlet prepare an external load balancer or the `metallb` module. The `HostPort` inlet is available for ClusterALBInstance only and does not require MetalLB.

Enabling the module and creating a Gateway

Enable the `alb` module as described in Enabling the module.

Create a ClusterALBInstance or ALBInstance resource. This creates and configures the managed Gateway object.

> ⚠ Manual modification of Gateway objects managed by the module is not allowed.

- ClusterALBInstance
- ALBInstance

Example of a ClusterALBInstance manifest for creating a cluster-wide gateway:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

Example of a minimal working ALBInstance configuration for publishing a gateway in a namespace:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

After the ALBInstance reaches the `Ready` state, create ListenerSet and HTTPRoute objects in the same namespace. Next steps are in “Publishing an application with ListenerSet and HTTPRoute” .

Creating a ListenerSet

The ListenerSet object describes system and user traffic handlers: hostname, TLS mode, port, and protocol. Each ListenerSet is linked to a parent Gateway through `spec.parentRef`. Routes are then attached to it.

Placement of ListenerSet objects depends on the type of Gateway object in use:

- For ClusterALBInstance, ListenerSet objects may be placed in any namespace.
- For ALBInstance, ListenerSet objects must be placed in the same namespace as the parent ALBInstance.

In both cases, place the ListenerSet in the same namespace as the associated HTTPRoute, GRPCRoute, and TLSRoute objects. In that case, you do not need additional setup such as creating a ReferenceGrant.

In ListenerSet, use ports `80` and `443` for HTTP/HTTPS. These are Gateway API listener ports. They are not the same as the `hostPort.httpPort` and `hostPort.httpsPort` parameters of the HostPort inlet, which set ports on the node.

TCPRoute and UDPRoute objects that use TCP and UDP ports from `additionalPorts` attach directly to the corresponding Gateway listener.

An example of a ListenerSet manifest for managing the reception of incoming HTTP and HTTPS requests through a cluster-wide gateway:

```
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: prod
spec:
  parentRef:
    name: public-gw  # The name of the Gateway object from the ClusterALBInstance
status, provided by the administrator.
    namespace: d8-alb
  listeners:
    - name: app-http
      port: 80 # In ListenerSet, use port 80 for HTTP. This is not the HostPort inlet
httpPort on the node.
      protocol: HTTP
      hostname: app.example.com
    - name: app-https
      port: 443 # In ListenerSet, use port 443 for HTTPS. This is not the HostPort
inlet httpsPort on the node.
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls   # Reference to the secret with the TLS certificate.
            namespace: prod
```

Creating routes

The following route types are used to route incoming requests:

- HTTPRoute: For routing HTTP/HTTPS/TLS requests. HTTPRoute objects support extended settings through annotations that complement the current Gateway API specification.
- GRPCRoute: For routing gRPC traffic.
- TLSRoute: For TLS passthrough routing.
- TCPRoute: For routing TCP traffic. For TCP ports from `additionalPorts`, attach TCPRoute directly to the Gateway listener, not to a ListenerSet.
- UDPRoute: For routing UDP traffic. For UDP ports from `additionalPorts`, attach UDPRoute directly to the Gateway listener, not to a ListenerSet.

An example of creating an HTTPRoute for HTTP and HTTPS traffic, and other publishing scenarios, are in “Publishing an application with ListenerSet and HTTPRoute” .

Publishing service domains

⚠ If you need to publish service domains, set the global parameter `publicDomainTemplate` — without it, system HTTPRoute/Gateway/ListenerSet objects for service domains will not work correctly, and the web interfaces of DP service components and other modules will not be published. If you do not need to publish service domains, you can leave this parameter unset.

To provide access to the DP cluster’s service domains, specify a default gateway. Create a ClusterALBInstance with the desired inlet type and configuration, and set `spec.defaultDeckhouseGateway: true` on it.

Example of a ClusterALBInstance manifest with `spec.defaultDeckhouseGateway: true`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  defaultDeckhouseGateway: true
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

After applying the changes, check the ClusterALBInstance status:

```
d8 k get clusteralbinstances
```

The ClusterALBInstance must reach the `Ready` state and create the managed Gateway. After that, ListenerSet and HTTPRoute objects appear in the corresponding system namespaces.

> **i** Currently, not all DP modules are available through the Gateway API. Do not disable the `ingress-nginx` module or delete related objects until the required web interfaces are published through the Gateway API and validated.

After you configure the default gateway, run the following command to see which modules have **already** published service HTTPRoute objects through the Gateway API in this cluster. The command shows the actual route inventory, not a full platform capability matrix:

```
d8 k get httproutes -A -l heritage=deckhouse -o json \
  | jq -r '
["MODULE","GATEWAY API SUPPORT"],
(.items
  | map(.metadata.labels.module // "UNKNOWN")
  | unique[]
  | ["d8-" + ., "yes"])
| @tsv
' \
  | column -t -s $'\t'
```

Selecting the default DP gateway when using multiple ClusterALBInstances

A cluster can have multiple cluster-scoped gateways simultaneously, each marked as the default gateway (with the `spec.defaultDeckhouseGateway: true` flag set for the corresponding ClusterALBInstance). In that case, the default gateway is the Gateway object created by the ClusterALBInstance object with the earliest `creationTimestamp` (that is, the one created first).

If no ClusterALBInstance object is marked as the default gateway, DP allows the Gateway object created by the `alb` module for the instance named `main` to be used as the default gateway.

Changing the default DP gateway

If DP system domains need to move to another Gateway object, complete these steps:

1. Create a new ClusterALBInstance object that describes the required settings and set `spec.defaultDeckhouseGateway: true` on it.
2. In the current ClusterALBInstance object that provides the default DP gateway, set `spec.defaultDeckhouseGateway: false`.
3. Check that all system ListenerSet objects now point to the new Gateway object in `spec.parentRef`.

Examples for different environments

This section describes traffic reception in different environments and operations on an already deployed ALB. Full CR field reference: ClusterALBInstance and ALBInstance. Additional examples are also

available in the `alb` module examples.

- Cloud provider
- Bare metal with MetalLB
- Bare metal with HostPort
- External L7 balancer

Example ClusterALBInstance with the `LoadBalancer` inlet for a cloud provider:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

To configure the cloud load balancer, specify the required Service annotations in `spec.inlet.loadBalancer.serviceAnnotations`. For example, for a Network Load Balancer in AWS:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer:
      serviceAnnotations:
        service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

To accept traffic on bare metal using the MetalLB load balancer, enable the `metallb` module and create a MetalLoadBalancerClass object with an address pool. Place MetalLB balancers on the same nodes as the Envoy Proxy pods of the `alb` module (typically frontend nodes labeled `node-role.deckhouse.io/frontend`):

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: alb
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  l2:
    # Network interfaces that announce the assigned IP addresses.
    interfaces:
      - eth1
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  type: L2
```

Then create a ClusterALBInstance object and set `spec.inlet.loadBalancer.loadBalancerClass`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer:
      loadBalancerClass: alb
      serviceAnnotations:
        # Number of addresses allocated from the pool declared in
MetalLoadBalancerClass.
        network.deckhouse.io/l2-load-balancer-external-ips-count: "1"
```

Example ClusterALBInstance with the `HostPort` inlet for bare metal without an external load balancer:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: HostPort
    hostPort:
      httpPort: 80
      httpsPort: 443
```

To place Envoy Proxy pods only on dedicated nodes, set `nodeSelector` and `tolerations` on the ClusterALBInstance.

If the `alb` module runs behind an external L7 balancer (for example, Cloudflare or Qrator), enable `useProxyProtocol` to receive real client addresses. Additionally, use `spec.originalIPDetection` to restrict the list of subnets allowed to provide headers with the client address.

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: HostPort
    hostPort:
      httpPort: 80
      httpsPort: 443
  useProxyProtocol: true
  originalIPDetection:
    setRealIPFrom:
      - 10.0.0.0/16
```

⚠ Proxy Protocol and HTTP/3 cannot be enabled at the same time: if both parameters are set, the `alb` module controller rejects the configuration as conflicting.

Changing the inlet while keeping the current Gateway

To change the inlet used for an existing Gateway object, complete these steps:

1. Create a new ClusterALBInstance object or ALBInstance object with a different name but the same `spec.gatewayName`, using the required inlet type.

2. Check that the new traffic path works correctly.
3. Delete the obsolete ClusterALBInstance object or ALBInstance object.

Because `gatewayName` does not change, the Gateway object stays the same. In most cases, the ListenerSet object and routes do not need to be rewritten.

Opening an additional TCP/UDP port on the cluster-wide gateway

If a dedicated TCP/UDP port is needed in addition to the default HTTP/HTTPS listeners, add the `spec.inlet.additionalPorts` field to the corresponding ClusterALBInstance object, for example:

```
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
    additionalPorts:
      - port: 9000
        protocol: TCP
```

The controller adds a corresponding TCP/UDP traffic handler to the managed Gateway object with a section name (`sectionName`) like `tcp-port-9000`. To attach a TCPRoute to it, specify the Gateway name and the matching `sectionName` in the route:

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: app-tcp
  namespace: prod
spec:
  parentRefs:
    - name: public-gw
      namespace: d8-alb
      sectionName: tcp-port-9000
      port: 9000
  rules:
    - backendRefs:
        - name: tcp-svc
          port: 9000
```

> **i** If a TCPRoute or UDPRoute object is created in a namespace different from the Gateway object namespace, create a ReferenceGrant object in the Gateway's namespace that allows references from the route's namespace.

UDPRoute examples and application publishing steps are in "Working with GRPCRoute, TLSRoute, TCPRoute, and UDPRoute objects".

Separating public and administrative zones

You can run separate Gateway objects for public and administrative traffic. This lets you assign each zone its own entry point and access policy. This approach is similar to using a separate Ingress NGINX Controller and IngressClass for the administrative zone.

Create a dedicated Gateway object for each zone and restrict administrative traffic with `spec.acceptRequestsFrom`. The decision is based on the real connection address, not on request headers.

Example of creating two ClusterALBInstance objects — for the public and administrative zones:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
---
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: admin
spec:
  gatewayName: admin-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
```

Then create separate ListenerSet objects and routes for each gateway. Application publishing examples are in “Publishing an application with ListenerSet and HTTPRoute” .

Namespaced publishing (ALBInstance)

Example of a minimal working ALBInstance configuration for publishing a gateway in a namespace:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

After the ALBInstance reaches the `Ready` state, create ListenerSet and HTTPRoute objects in the same namespace. Next steps are in “Publishing an application with ListenerSet and HTTPRoute” .

Issuing TLS certificates with cert-manager

The `alb` module uses `cert-manager` to automatically issue TLS certificates. The `d8-http` and `d8-https` listeners serve HTTP-01 challenges when issuing a certificate. To issue a certificate for an application, create a Certificate object that stores the certificate in a Secret, and reference that Secret from `certificateRefs` in the corresponding ListenerSet.

Certificate example that creates the `app-tls` Secret for a ListenerSet:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: app-tls
  namespace: prod
spec:
  secretName: app-tls
  issuerRef:
    name: letsencrypt
    kind: ClusterIssuer
  dnsNames:
    - app.example.com
```

Certificates are issued by the `cert-manager` module. If the `alb` module is used alongside `ingress-nginx`, use a separate ClusterIssuer for each ALB type, as described in “Alongside ingress-nginx”.

Using with other modules and third-party solutions

ALBs implemented using the Kubernetes Gateway API in a DP cluster can be used in conjunction with ALBs implemented using the Ingress NGINX Controller, as well as with ALBs based on third-party Gateway API solutions. For a step-by-step cutover, see Migrating from ingress-nginx to alb.

Alongside ingress-nginx

ALBs implemented using the Kubernetes Gateway API can run in a cluster alongside “ALB with Ingress NGINX Controller”. In that case, use a separate ClusterIssuer object for each ALB type so that certificate settings and lifecycles are managed independently. The same external hostname must not be served by both ALBs at once without an explicit split at the DNS or external load balancer layer.

> **i** For the DP gateway, a ClusterIssuer object is automatically created by default. This same ClusterIssuer object is used to issue certificates for system domains.

Alongside third-party Gateway API implementations

Use of third-party Gateway API implementations is supported, provided that the cluster uses the following Gateway API object storage versions compatible with the `alb` module controller:

- BackendTLSPolicy: v1;
- GatewayClass: v1;
- Gateway: v1;
- ListenerSet: v1;
- GRPCRoute: v1;
- HTTPRoute: v1;
- ReferenceGrant: v1beta1;
- TCPRoute: v1alpha2/v1;
- UDPRoute: v1alpha2/v1;
- TLSRoute: v1.

During startup, the `alb` module controller checks the currently stored versions of Gateway API objects. If it detects a mismatch between the installed and required versions, it stops and does not proceed. If a given Gateway API object type is completely absent from the cluster, the controller automatically creates the required CRD version and then continues startup.

To manually verify version compatibility of the installed Gateway API objects in the cluster, use:

```
declare -A want=(
    [gatewayclasses.gateway.networking.k8s.io]=v1
    [gateways.gateway.networking.k8s.io]=v1
    [grpcroutes.gateway.networking.k8s.io]=v1
    [httproutes.gateway.networking.k8s.io]=v1
    [listenersets.gateway.networking.k8s.io]=v1
    [referencegrants.gateway.networking.k8s.io]=v1beta1
    [tcproutes.gateway.networking.k8s.io]="v1|v1alpha2"
    [udproutes.gateway.networking.k8s.io]="v1|v1alpha2"
    [tlsroutes.gateway.networking.k8s.io]=v1
    [backendtlspolicies.gateway.networking.k8s.io]=v1
)

for crd in "${!want[@]}"; do
    got="$(
       d8 k get crd "$crd" \
         -o jsonpath='{.spec.versions[?(@.storage==true)].name}' \
         2>/dev/null || true
    )"
    if [[ -n "$got" && "$got" =~ ^(${want[$crd]})$ ]]; then
         echo "$crd OK storage=$got"
    else
         echo "$crd FAILED cluster=${got:-MISSING} expected=${want[$crd]}"
    fi
done | sort
```

Otherwise, the module only configures and manages Gateway objects associated with its designated GatewayClass, which minimizes the risk of conflicts when third-party Gateway API implementations are present.

Enabling HTTP/3

By default the gateway accepts HTTP/1.1 and HTTP/2. To enable HTTP/3, set `spec.enableHTTP3: true` on the ClusterALBInstance or ALBInstance:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  enableHTTP3: true
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

⚠ You cannot enable `enableHTTP3` and `useProxyProtocol` at the same time: if both parameters are set, the `alb` module controller rejects the configuration as conflicting.

Keep the following in mind:

- HTTP/3 uses QUIC over UDP. Make sure the load balancer and network rules allow UDP on the HTTPS port (usually `443`), not only TCP.
- For the `LoadBalancer` inlet, confirm that the cloud or MetalLB load balancer accepts UDP on the required port.
- Clients must support HTTP/3. From a workstation, verify with:

```
curl -vk --http3 https://app.example.com/
```

If `curl` was built without HTTP/3, use a QUIC-capable client or check UDP port reachability and Envoy Proxy logs.

Using GeoIP and GeoLite2

The `alb` module supports adding GeoIP fields to HTTP request headers based on MaxMind GeoIP/GeoLite2 databases.

Currently, the following database editions can be used:

- GeoIP2-Anonymous-IP;
- GeoIP2-City;
- GeoIP2-ISP;
- GeoIP2-ASN;
- GeoLite2-ASN;
- GeoLite2-City.

> **i** The current GeoIP integration supports using up to 4 databases simultaneously.

Choose how GeoIP databases are obtained:

- Downloading databases from MaxMind
- Local mirror

To use GeoIP and download databases directly from MaxMind servers, first create a Secret containing the license key, for example:

```
d8 k -n prod create secret generic geoip-license \
  --from-literal=licenseKey='<MAXMIND_LICENSE_KEY>'
```

> **i** When configuring GeoIP for ClusterALBInstance, the Secret can be placed in any namespace, but it is recommended to use `d8-alb`.
>
> For ALBInstance objects, the Secret must reside in the same namespace as the ALBInstance object.

After creating the Secret, reference it in the `spec.geoIP.licenseKeySecretRef` parameter of a ClusterALBInstance or ALBInstance object, for example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    licenseKeySecretRef:
      name: geoip-license
```

To use GeoIP and download databases from a local mirror, specify the mirror URL in `spec.geoIP.maxmindMirror.url`, for example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    maxmindMirror:
      url: "https://local.geoip:8443"
```

You can also use a URL pointing to a local caching GeoIP server in another namespace, for example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    maxmindMirror:
      url: "http://geoproxy-cluster.d8-alb.svc:8080/download"
```

Using GeoIP Headers

Once GeoIP is configured in the namespace where ClusterALBInstance or ALBInstance proxies reside, a caching and update server for GeoIP databases is started. Envoy Proxy pods are then restarted sequentially so they can fetch GeoIP databases from the local GeoIP server.

To add GeoIP fields to HTTP request headers, specify the names of the HTTP headers in `spec.geoIP.headers`, for example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    headers:
      city: geoip_city
      country: geoip_country
    licenseKeySecretRef:
      name: geoip-license
    maxmindEditionIDs:
      - GeoLite2-City
```

GeoIP databases are updated once per day, both on the caching server and in each individual Envoy Proxy pod using the caching server.

PVC settings for GeoIP components are controlled by the `storageClass` module parameter.

Configuring OpenTelemetry tracing

The `alb` module supports exporting OpenTelemetry traces from Envoy Proxy.

To enable export, set the OpenTelemetry Collector endpoint in `spec.openTelemetry.tracing.url` — using the `http://`, `https://`, or `grpc://` scheme, with the host, port, and, if needed, a path (for example, for OTLP/HTTP).

The `randomSamplingPercentage` parameter sets the share of requests that Envoy Proxy randomly selects for tracing (25% by default).

When using TLS, explicitly set the `sni` parameter if the OpenTelemetry Collector is behind a proxy or load balancer that selects upstreams based on Server Name Indication.

Configure TLS in `spec.openTelemetry.tracing.tls`.

Configuring OpenTelemetry tracing TLS

If OpenTelemetry tracing must send data over TLS, create a Kubernetes Secret with the CA certificate and reference it from `spec.openTelemetry.tracing.tls.caSecretName`.

For ClusterALBInstance and the default DP gateway, place the Secret in the `d8-alb` namespace. For

ALBInstance, place the Secret in the same namespace as the ALBInstance object. The Secret must contain the `cacert` key. Additional Subject Alternative Names for verifying the OpenTelemetry Collector certificate are set with `subjectAltNames`, and `insecureSkipVerify` disables this verification.

```
apiVersion: v1
kind: Secret
metadata:
  name: otel-tracing-ca
  namespace: d8-alb
type: Opaque
stringData:
  cacert: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
---
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: proxy-gw
spec:
  gatewayName: proxy-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
  openTelemetry:
    tracing:
      url: https://otel-collector.monitoring.svc.cluster.local:4318/v1/traces
      tls:
        sni: otel-collector.monitoring.svc.cluster.local
        caSecretName: otel-tracing-ca
```

Diagnostics and verification

Use this section to check that the gateway is ready and to answer common first-setup questions. More scenarios are in the “alb module FAQ”.

- After creating a ClusterALBInstance or ALBInstance, wait for the `Ready` state and take the Gateway name and namespace from `status`. Field descriptions are in ClusterALBInstance status and ALBInstance status:

  ```
  d8 k get clusteralbinstance
  d8 k -n <NAMESPACE> get albinstance
  d8 k -n d8-alb get gateway
  ```

- With the LoadBalancer inlet, get the address from the load balancer Service (usually in the `d8-alb` namespace) and point DNS for the ListenerSet hostname to it:

  ```
  d8 k -n d8-alb get svc
  ```

- Check whether the same hostname is used for two ALBs. The `alb` and `ingress-nginx` modules can run in the same cluster, but the same external hostname must not be served by both ALBs at once without an explicit split at the DNS or external load balancer layer.
- Check for a conflict between ListenerSet objects. If two ListenerSet objects with identical handlers point to the same Gateway, the controller rejects the conflicting configuration. Change the hostname, port, or protocol, or remove the duplicate ListenerSet.
- For troubleshooting, you can inspect the configuration that the controller and the proxy configurator pushed into the Envoy Proxy instance that serves the Gateway object.

  1. Select an Envoy Proxy pod for the required Gateway object:

```
d8 k -n d8-alb get pods -l alb.deckhouse.io/gateway=shared-gateway
```

2. Get the configuration through the following command (replace `<ENVOY_PROXY_POD_NAME>` with the Envoy Proxy pod name from the previous step):

   ```
   d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump
   ```

   If only one section of the configuration is needed, the required section may be requested explicitly:

   ```
   d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
   resource=dynamic_listeners
   d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
   resource=dynamic_route_configs
   d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
   resource=dynamic_active_clusters
   ```

This makes it easy to check whether the expected traffic handlers, virtual hosts, and upstream clusters appeared after changes to the ListenerSet object or Route object.

Additional resources

- Migrating from ingress-nginx to alb
- Publishing applications with Kubernetes Gateway API
- Incoming traffic balancing
- Module `alb` documentation
- Global parameters
- Module `alb` custom resources
- Module `alb` parameters
- Module `ingress-nginx` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Migrating from ingress-nginx to alb

This guide describes migration from the `ingress-nginx` module to the `alb` module. Within this migration, application publishing transitions from the Ingress API to the Gateway API.

The guide covers model differences, preparing `alb` module infrastructure with ClusterALBInstance or ALBInstance and a managed Gateway, migrating applications to the Gateway API, migrating Deckhouse Platform (DP) system interfaces, switching traffic to the `alb` module, and rolling back if needed.

Procedure:

1. Preparing alb module infrastructure.
2. Migrating DP interfaces — only if system interfaces currently use Ingress and must move to the Gateway API.
3. Migrating application publishing.
4. Switching traffic to the alb module.
5. Cleanup.

Reasons to move to the Gateway API

Main reasons to move from the Ingress API to the Gateway API:

- Active maintenance of the upstream Ingress NGINX project used by DP has ended. Further upstream development of features, fixes, and integrations is no longer expected. For new application publishing scenarios, use the Gateway API.
- Unlike the Ingress API, the Gateway API describes routes with protocol-specific resources, configures traffic entry points explicitly, controls route attachment, and manages cross-namespace access through dedicated resources. Complex traffic configurations can be defined

with API resources instead of relying mainly on controller-specific annotations.
- The Gateway API separates responsibilities by role. Cluster and network administrators manage traffic infrastructure and Gateway objects through ClusterALBInstance or ALBInstance. Namespace administrators configure traffic reception through ListenerSet objects (hostname, TLS, ports). Application developers define routing with HTTPRoute and other route resources. This separation supports delegated configuration, validation, and gradual migration.

Model comparison

This section compares how administrators define traffic infrastructure, how DP provisions it, and how application routing configuration is translated into data-plane configuration.

Ingress API and ingress-nginx

The diagram below shows the resource model and traffic flow through the `ingress-nginx` module.

HTTP and HTTPS traffic processing through the `ingress-nginx` module is configured as follows:

1. A cluster administrator creates a cluster-scoped IngressNginxController object.
2. The IngressNginxController specifies the name of the IngressClass it uses. If the name is omitted, `nginx` is used.
3. DP reconciles the object and provisions the required infrastructure, including the IngressClass. Multiple IngressNginxController objects can use the same IngressClass.
4. By default, DP resources are published through the IngressClass named `nginx`. A different class can be selected in the DP global configuration.
5. Network administrators or application development teams create Ingress objects that select the required IngressClass explicitly or implicitly.
6. The resulting nginx configuration combines infrastructure settings from IngressNginxController with the Ingress objects selected by the IngressClass.

Gateway API and alb

The diagram below shows the resource model and traffic flow through the `alb` module.

HTTP, HTTPS, gRPC, TLS, TCP, and UDP traffic processing through the `alb` module is configured as follows:

1. A cluster administrator creates a cluster-scoped ClusterALBInstance object.
2. The ClusterALBInstance specifies infrastructure parameters and the mandatory `gatewayName`, which identifies the managed Gateway.
3. The `alb` module controller reconciles the object and provisions the managed Gateway and the required traffic-processing infrastructure. Multiple ClusterALBInstance objects can use the same `gatewayName` and therefore the same Gateway.

4. If a default DP Gateway is configured, DP modules create their ListenerSet, HTTPRoute, and other Gateway API resources for that Gateway.
5. Network administrators or application development teams create Gateway API resources and attach them to the managed Gateway.
6. The resulting Envoy Proxy configuration combines infrastructure settings from ClusterALBInstance with the configuration represented by the Gateway API resources attached to the Gateway.

Key architectural differences

The table highlights the following architectural differences:

Parameter	Ingress API and ingress-nginx	Gateway API and alb
Resource relationships	Infrastructure and routing rules are associated indirectly through an IngressClass	Resources form an explicit graph using `parentRefs` and other typed references, with the Gateway as the root
Separation of responsibilities	IngressNginxController defines infrastructure, while Ingress combines application routing with controller-specific configuration	Cluster and network administrators manage instances and gateways, namespace administrators manage ListenerSet objects, and application developers manage HTTPRoute and other route resources
Configuration model	An Ingress combines most HTTP/HTTPS routing configuration in one object and controller-specific annotations	Listeners, routes, backends, and policies are represented by separate, composable objects
Shared configuration and entry points	Multiple IngressNginxController objects that use the same IngressClass consume the same set of Ingress rules while providing separate entry points	Multiple ClusterALBInstance objects with the same `gatewayName` provide separate entry points backed by the same Gateway configuration
Cross-namespace references	An Ingress, its backend Service, and its TLS Secret generally reside in the same namespace	Controlled cross-namespace references are supported through ReferenceGrant
Protocol support	Ingress primarily models HTTP and HTTPS traffic	Dedicated route types model HTTP, gRPC, TCP, TLS, and UDP traffic
Extensibility	Additional behavior is commonly configured through implementation-specific annotations	More behavior is expressed through structured, validated API resources and policies
Lifecycle and ownership	Infrastructure and routing configuration have limited delegation and attachment controls	Gateway infrastructure can remain stable while application development teams independently create, update, and remove routes
Route attachment	Selecting an IngressClass provides a broad association between an Ingress and controllers	Gateway objects and listeners explicitly control which routes may attach to them

⚠ Multiple ClusterALBInstance objects can refer to the same Gateway, but Gateway-level settings such as `additionalPorts` can conflict. Use compatible Gateway-level settings on every instance associated with the same Gateway.

Consistent with the Gateway API conflict-resolution model, the `alb` module builds the final configuration from the oldest ClusterALBInstance. Conflicting settings from newer instances are ignored and reported in their status.

Step 1. Preparing alb module infrastructure

On this step, choose the `alb` module instance type (ClusterALBInstance or ALBInstance), configure the inlet, and prepare TLS certificates for the Gateway API entry point.

Choosing ALBInstance or ClusterALBInstance

When choosing the instance type, keep the following in mind:

- Use ClusterALBInstance for a shared or platform-level Gateway, for publishing DP system interfaces, or when the `HostPort` inlet is required.
- To publish DP system interfaces, follow Publishing service domains.
- Use ALBInstance for a Gateway dedicated to an application or team and managed within its namespace. ALBInstance supports the `LoadBalancer` and `ClusterIP` inlets; for migration from `ingress-nginx`, `LoadBalancer` is the relevant one.
- A detailed comparison is in ClusterALBInstance and ALBInstance.

Inlet configuration

In IngressNginxController, the inlet type combines the method of accepting traffic with optional behaviors such as Proxy Protocol and SSL passthrough. The `alb` module configures these concerns separately:

- ClusterALBInstance supports the `LoadBalancer` and `HostPort` inlet types.
- ALBInstance supports the `LoadBalancer` and `ClusterIP` inlet types.
- Proxy Protocol is enabled by the `spec.useProxyProtocol` parameter. It can be enabled or disabled on an existing instance without restarting the Envoy Proxy Pod objects or recreating the instance.
- TLS passthrough is configured with a TLS listener and a TLSRoute, rather than with a dedicated inlet type.

Use the following mapping when selecting an inlet for the `alb` module:

IngressNginxController inlet	alb module configuration	Migration notes
`LoadBalancer`	ClusterALBInstance or ALBInstance with `spec.inlet.type: LoadBalancer`	The controller provisions a Service of type `LoadBalancer`
`LoadBalancerWithProxyProtocol`	`LoadBalancer` inlet with `spec.useProxyProtocol: true`	Configure the external load balancer to send Proxy Protocol. Proxy Protocol and HTTP/3 cannot be enabled simultaneously: the `alb` module controller rejects such a configuration as conflicting
`LoadBalancerWithSSLPassthrough`	`LoadBalancer` inlet with a TLS listener and TLSRoute	TLS passthrough is part of the Gateway API routing configuration and is not an inlet variant
`HostPort`	ClusterALBInstance with `spec.inlet.type: HostPort`	`HostPort` is not supported by ALBInstance
`HostPortWithProxyProtocol`	ClusterALBInstance with the `HostPort` inlet and `spec.useProxyProtocol: true`	Proxy Protocol and HTTP/3 cannot be enabled simultaneously
`HostPortWithSSLPassthrough`	ClusterALBInstance with the `HostPort` inlet, a TLS listener, and TLSRoute	TLS passthrough is configured independently of the inlet
`HostWithFailover`	No direct equivalent	Use a ClusterALBInstance with the `LoadBalancer` inlet backed by MetalLB. Follow “Example for bare metal with the MetalLB load balancer” and validate load-balancer failover before switching traffic

> ⚠ During migration, the `ingress-nginx` module with a `HostPort` or `HostWithFailover` inlet and the `alb` module with a `HostPort` inlet cannot use the same host ports on the same nodes. Select separate node sets or a different set of host ports for one of the controllers.

Map related inlet parameters as follows:

IngressNginxController	ClusterALBInstance or ALBInstance
`spec.loadBalancer.annotations`	`spec.inlet.loadBalancer.serviceAnnotations`
`spec.loadBalancer.loadBalancerClass`	`spec.inlet.loadBalancer.loadBalancerClass`
`spec.loadBalancer.httpPort`, `httpsPort`	`spec.inlet.loadBalancer.httpPort`, `httpsPort`
`spec.loadBalancer.sourceRanges`	`spec.inlet.loadBalancer.loadBalancerSourceRanges`
`spec.hostPort.httpPort`, `httpsPort`	`spec.inlet.hostPort.httpPort`, `httpsPort`
`spec.acceptRequestsFrom`	`spec.acceptRequestsFrom`
`spec.*.behindL7Proxy`, `realIPHeader`, `acceptClientIPHeadersFrom`	`spec.originalIPDetection.realIPHeader`, `setRealIPFrom`

When migrating these parameters, consider the following:

- Copy only service annotations supported by the target load balancer implementation.
- Set `spec.inlet.loadBalancer.loadBalancerClass` at creation — the parameter is immutable afterward.
- Keep the default HTTP and HTTPS ports as `80` and `443`, or set a port to `0` in the `alb` module to disable the corresponding default listener.
- Configure HostPort parameters for ClusterALBInstance only. Specify at least one port.
- Preserve the required source CIDR restrictions in `spec.acceptRequestsFrom`.
- Configure trusted proxy CIDRs explicitly. Do not trust client-IP headers from arbitrary sources.
- Verify `loadBalancerSourceRanges` behavior with the target load balancer implementation. The value is passed to the `LoadBalancer` Service. Cloud providers may not support or may ignore this parameter.

The inlet type is immutable for both ClusterALBInstance and ALBInstance. To change the inlet, create a new instance with the same `gatewayName`, validate traffic, switch to it, and delete the old instance.

TLS and certificates

> ⚠ When `ingress-nginx` and `alb` are used simultaneously and certificates are issued by Issuer or ClusterIssuer resources with HTTP-01 solvers, use separate Certificate resources and Secret objects for the Ingress API and Gateway API paths. Otherwise certificate issuance or renewal may conflict.

DP provisions a Gateway-specific ClusterIssuer with a Let’s Encrypt HTTP-01 solver for the default DP Gateway. Separate Certificate and Secret objects are required only for Issuer and ClusterIssuer resources with HTTP-01 solvers. This does not apply to resources configured exclusively with DNS-01 solvers.

Instructions for configuring an HTTP-01 issuer with the Gateway API solver are in “Adding a custom HTTP-01 ClusterIssuer or Issuer for ALB”.

To finish preparing the infrastructure, do the following:

1. Create a ClusterALBInstance or ALBInstance with the selected inlet and the parameters from the tables above.
2. Wait until the instance and the managed Gateway are ready.
3. Prepare separate Certificate resources and Secret objects for the Gateway API path when Issuer or ClusterIssuer resources with HTTP-01 solvers are used at the same time.

Step 2. Migrating DP interfaces

If DP system interfaces are published through Ingress and must move to the Gateway API, follow Publishing service domains.

> **i** Not every DP module publishes service HTTPRoute objects through the Gateway API yet. The `jq` command from that section shows which modules already do this in your cluster — it lists the routes actually published, not the full set of capabilities the platform supports.

If you do not need to migrate DP interfaces, continue with step 3.

Step 3. Migrating application publishing

The list of Gateway API resources supported by the `alb` module and ownership details for managed Gateway objects are in “Supported Gateway API resources”.

Converting Ingress to Gateway API

To convert Ingress resources to Gateway API resources, do the following:

1. Confirm that the target ALBInstance or ClusterALBInstance was created in step 1 and is ready.
2. Obtain the managed Gateway namespace and name from the instance status.
3. Convert each Ingress and its related configuration into the appropriate ListenerSet, route, and policy resources for that Gateway. The built-in conversion tool produces a draft, but not every `ingress-nginx` feature has a direct Gateway API equivalent.
4. Review and adjust the generated manifests before applying them.
5. Apply the resources:

   ```
   d8 k apply -f gateway-api.yaml
   ```

6. Verify Gateway, ListenerSet, and route statuses before switching traffic.

Using the built-in ingress2gateway tool

The gateway controller provides an HTTP endpoint that accepts a single object, a Kubernetes List object, or a YAML payload containing multiple documents. The converter reads the following input resources:

- Ingress — the resources selected for conversion.
- Service — used to resolve named backend ports and service metadata.
- DexAuthenticator — used to resolve `spec.applicationDomain` when an `ingress-nginx` external-auth configuration contains nginx variables such as `$host`.

Pass all related resources as input so the converter can preserve the supported parts of their configuration. The endpoint is disabled by default and listens only inside gateway controller Pod objects.

1. Temporarily enable `migrations.ingress2Gateway.enabled` in the `alb` module configuration and wait for the gateway controller to restart:

   ```
   d8 k patch moduleconfig alb --type merge \
     --patch '{"spec":{"settings":{"migrations":{"ingress2Gateway":
   {"enabled":true}}}}}'
   d8 k -n d8-alb rollout status deployment/gateway-controller
   ```

2. Forward the endpoint from a gateway controller Pod:

   ```
   d8 k -n d8-alb port-forward deployment/gateway-controller 8082:8082
   ```

3. In another terminal, export all recognized resource types and send the resulting Kubernetes List directly to the converter:

```
converter_url='http://127.0.0.1:8082/ingress2gateway'
converter_url="${converter_url}?gateway=<GATEWAY_NAMESPACE>/<GATEWAY_NAME>"
converter_url="${converter_url}&scope=<SCOPE>&ingress-class=<INGRESS_CLASS>"

d8 k get ingress,service,dexauthenticator \
  --all-namespaces --output yaml | \
  curl --fail-with-body --silent --show-error --request POST \
    --header 'Content-Type: application/yaml' \
    --data-binary @- \
    --output gateway-api.yaml \
    "$converter_url"
```

In the URL, query parameter names are lowercase: `gateway`, `scope`, and `ingress-class`. In the example, their values are placeholders in angle brackets and uppercase:

- `<GATEWAY_NAMESPACE>` — namespace of the managed Gateway from the instance status in step 1;
- `<GATEWAY_NAME>` — name of the managed Gateway from the same status.

  The `<GATEWAY_NAMESPACE>/<GATEWAY_NAME>` pair is the value of the `gateway` parameter. Use the default value `d8-alb/public-gw` only if it matches your Gateway.
- `<SCOPE>` — `cluster` for the cluster-scoped model, or `namespaced` for a Gateway in an application namespace (as with ALBInstance). If the parameter is omitted, `cluster` is used;
- `<INGRESS CLASS>` — IngressClass of the source Ingress resources. If the parameter is omitted, `nginx` is used.

> **i** The request body is limited to 8 MiB. For large clusters, export only the namespaces and related resources being migrated.

4. Review `gateway-api.yaml` and the conversion diagnostics included as YAML comments, then apply the resources.
5. After completing the conversion, disable the endpoint:

```
d8 k patch moduleconfig alb --type merge \
  --patch '{"spec":{"settings":{"migrations":{"ingress2Gateway":
{"enabled":false}}}}}'
```

Extending Gateway API with annotations

The Gateway API specification does not cover every implementation-specific traffic-management feature required by DP. The `alb` module therefore uses HTTPRoute annotations for options that are not yet represented by standard Gateway API fields.

As the corresponding features become available in the Gateway API, the `alb` module will gradually replace annotation-based configuration with native Gateway API resources and fields.

During migration, use standard Gateway API fields where possible and replace `ingress-nginx` annotations only with supported `alb` annotations — including after you apply `gateway-api.yaml`. The current list is in “Supported HTTPRoute annotations”.

The `ingress2gateway` utility produces a draft of Gateway API resources and **does not** migrate every `ingress-nginx` annotation automatically. After conversion, compare the Ingress with the table below and apply the required settings manually.

Ingress annotation (`ingress-nginx`)	Equivalent in the `alb` module
`nginx.ingress.kubernetes.io/whitelist-source-range`	HTTPRoute annotation `alb.network.deckhouse.io/whitelist-source-range`
`nginx.ingress.kubernetes.io/service-upstream`	HTTPRoute annotation `alb.network.deckhouse.io/service-upstream`

Ingress annotation (`ingress-nginx`)	Equivalent in the `alb` module
`nginx.ingress.kubernetes.io/upstream-vhost`	HTTPRoute `URLRewrite` filter with `hostname` (see publishing with an Istio sidecar)
`nginx.ingress.kubernetes.io/auth-url` / `auth-signin`	HTTPRoute annotations `alb.network.deckhouse.io/auth-url` and `alb.network.deckhouse.io/auth-signin`
`nginx.ingress.kubernetes.io/auth-type: basic` and a Secret	HTTPRoute annotation `alb.network.deckhouse.io/basic-auth-secret`
`nginx.ingress.kubernetes.io/limit-rps`	HTTPRoute annotation `alb.network.deckhouse.io/limit-rps`
`nginx.ingress.kubernetes.io/proxy-body-size`	HTTPRoute annotation `alb.network.deckhouse.io/buffer-max-request-bytes` (value in bytes)
`nginx.ingress.kubernetes.io/proxy-buffer-size`	HTTPRoute annotation `alb.network.deckhouse.io/proxy-buffer-size`
`nginx.ingress.kubernetes.io/proxy-read-timeout` / `proxy-send-timeout`	HTTPRoute annotation `alb.network.deckhouse.io/idle-timeout` (idle timeout, not total request duration)
`nginx.ingress.kubernetes.io/affinity` / cookie	HTTPRoute annotation `alb.network.deckhouse.io/session-affinity`
`nginx.ingress.kubernetes.io/rewrite-target`	HTTPRoute annotation `alb.network.deckhouse.io/rewrite-target` or standard Gateway API filters
`nginx.ingress.kubernetes.io/configuration-snippet` and other nginx snippets	No direct equivalent; redesign the configuration for Gateway API and `alb` annotations

If an Ingress annotation is missing from the table and has no Gateway API field, behavior after migration may differ or disappear. Validate the application on the `alb` module path before switching production traffic.

Step 4. Switching traffic to the alb module

Run `ingress-nginx` and `alb` simultaneously until the `alb` module path has been validated and the rollback window has closed.

Migrate individual domains or namespaces when they can use separate DNS records. Otherwise, switch the shared external entry point by changing DNS records or the backend pool of an external load balancer.

On this step:

- Test the `alb` module without changing DNS with `migrationGateway` when you need validation without a DNS change (`ingress-nginx` version `1.1.0` and later).
- Enable `http01CertificateSolverBridging` when Gateway API path certificates use HTTP-01 while public DNS still points to Ingress.
- Choose the switching method for your current entry point in the subsections below: automatically provisioned load balancer, manually managed load balancer, or HostPort/HostWithFailover.
- Before the final production cutover, complete “Validating the cutover”.

Testing the alb module via the Ingress NGINX Controller

The `spec.migrationGateway` parameter of IngressNginxController enables source-IP-based testing without changing DNS.

Requests from `sourceCIDRs` continue to enter through the existing Ingress NGINX Controller entry point but are proxied to the internal Service of the target `alb` module instance. Other requests continue to use the original Ingress backends.

> **i** The parameter is available in the `ingress-nginx` module version `1.1.0` and later.

Despite its use in this guide for migration to the `alb` module, `migrationGateway` is not specific to the `alb` module.

Its `serviceRef` can point to the entry-point Service of any Gateway API implementation that accepts ordinary HTTP and HTTPS traffic on the configured ports.

Locate the `alb` module configuration Service:

```
d8 k get service --all-namespaces \
  --selector alb.deckhouse.io/configuration-service
```

Example output for a ClusterALBInstance named `main` — a `ClusterIP` Service in the `d8-alb` namespace:

```
NAMESPACE   NAME   TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
d8-alb      main   ClusterIP   10.222.1.10    <none>        80/TCP,443/TCP   10d
```

For an ALBInstance, the Service is in the ALBInstance's namespace, named `d8-alb-<ALB_INSTANCE_NAME>`.

Configure the source IngressNginxController, using narrow tester CIDRs initially:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <CONTROLLER_NAME>
spec:
  # Spec fields unrelated to migrationGateway (inlet, ingressClass, and so on) are
omitted.
  migrationGateway:
    sourceCIDRs:
      - <SOURCE_CIDR>
    serviceRef:
      namespace: <SERVICE_NAMESPACE>
      name: <SERVICE_NAME>
      ports:
        http: <HTTP_PORT>
        https: <HTTPS_PORT>
```

where:

- `<CONTROLLER_NAME>` — name of the source IngressNginxController;
- `<SOURCE_CIDR>` — CIDR of clients whose requests are proxied to the `alb` module during testing;
- `<SERVICE_NAMESPACE>` — namespace of the `alb` module configuration Service;
- `<SERVICE_NAME>` — name of the `alb` module configuration Service;
- `<HTTP_PORT>` — HTTP port on the target Service;
- `<HTTPS_PORT>` — HTTPS port on the target Service.

For matching clients, `migrationGateway` forwards requests to the target Service ports and bypasses the backends and location-level behavior from the original Ingress resources. HTTP requests go to the configured HTTP port. For HTTPS, nginx terminates the incoming TLS connection and establishes a new TLS connection to the configured HTTPS port, using the original hostname in the Host header and for SNI. HTTP, HTTPS, and HTTP-based protocol upgrades such as WebSocket are supported. gRPC and non-HTTP protocols are not.

The setting applies to every Ingress served by the controller. Before expanding `sourceCIDRs`, verify every hostname reachable from those addresses and confirm that authentication, redirects, headers, GeoIP behavior, WebSocket connections, and other required policies exist on the `alb` module path.

The `migrationGateway` parameter works by forwarding an already-accepted, and if needed re-encrypted, HTTP or HTTPS request to the target Service, so it is not supported where nginx does not perform this processing:

- With the `HostPortWithSSLPassthrough` and `LoadBalancerWithSSLPassthrough` inlets — with TLS passthrough, nginx does not decrypt traffic and cannot determine which request to forward.
- With the `HostWithFailover` inlet — this inlet always uses Proxy Protocol, and `migrationGateway` does not support sending Proxy Protocol to the target Service.
- With `enableIstioSidecar` — traffic is handled by the Istio sidecar rather than directly by nginx.

The target Service must accept ordinary HTTP and HTTPS without Proxy Protocol — `migrationGateway` does not send it.

If the `alb` module path needs Proxy Protocol in production, disable it for `migrationGateway` testing (`spec.useProxyProtocol: false`). Re-enable it before shifting production traffic and validate the entry point through the load balancer that sends Proxy Protocol.

When a trusted L7 proxy is used in front of the Ingress NGINX Controller, verify real-IP processing before relying on `sourceCIDRs`.

To immediately return selected clients to the original Ingress backends, remove `spec.migrationGateway`:

```
d8 k patch ingressnginxcontroller <CONTROLLER_NAME> --type json \
  --patch '[{"op":"remove","path":"/spec/migrationGateway"}]'
```

where `<CONTROLLER_NAME>` is the name of the source IngressNginxController.

Preserving HTTP-01 validation

The `migrationGateway` parameter handles selected application requests.

The `migrations.http01CertificateSolverBridging` parameter keeps cert-manager Gateway API HTTP-01 challenges reachable through the Ingress NGINX Controller while public DNS still points to it.

When Issuer or ClusterIssuer resources with HTTP-01 solvers are used, do the following:

1. Enable bridging before requesting certificates for the Gateway API path:

   ```
   d8 k patch moduleconfig alb --type merge \
     --patch '{"spec":{"settings":{"migrations":{"http01CertificateSolverBridging":
   {"enabled":true,"ingressClassName":"<INGRESS_CLASS>"}}}}}'
   d8 k -n d8-alb rollout status deployment/gateway-controller
   ```

   where `<INGRESS_CLASS>` is the IngressClass that currently receives public traffic on port `80`.

2. Keep the `ingress-nginx` module enabled. The `alb` module controller creates temporary Ingress resources for cert-manager solver HTTPRoute resources, allowing challenges to pass through the Ingress NGINX Controller to the solver Service.

3. Issue certificates for the Gateway API path before the traffic cutover. Test the prepared TLS configuration through `migrationGateway` or by connecting directly to the Gateway API entry-point address with a hostname override, for example using the curl `--resolve` option.

4. After DNS or the external load balancer sends public traffic to the Gateway API entry point, verify HTTP-01 issuance through the Gateway API path and disable bridging:

   ```
   d8 k patch moduleconfig alb --type merge \
     --patch '{"spec":{"settings":{"migrations":{"http01CertificateSolverBridging":
   {"enabled":false}}}}}'
   ```

Choosing the switching method

Choose the switching method based on how external traffic is accepted today: through an automatically provisioned load balancer, a manually managed load balancer, or direct HostPort/HostWithFailover access.

- DNS / provisioned LB
- Manually managed LB
- HostPort or HostWithFailover

Automatically provisioned load balancer

To switch traffic through DNS, do the following:

1. Wait for the `alb` module load balancer and its health checks to become ready.
2. Lower the DNS TTL in advance.
3. Change each application DNS record from the Ingress NGINX Controller address to the Gateway API entry-point address.

When the Ingress NGINX Controller uses a `LoadBalancer` Service provisioned by the Kubernetes cloud provider, the source and target controllers normally have separate load balancer IP addresses or hostnames.

Use weighted DNS records for a gradual switch only if the DNS provider supports them.

A provider-managed load balancer does not normally allow node-by-node migration between two independently managed Service objects.

If MetalLB or another implementation uses a fixed address, the same address cannot be assigned to both Service objects simultaneously. Transfer it only during a coordinated cutover.

Manually managed load balancer

To switch traffic through an external load balancer, do the following:

1. Keep the public DNS record unchanged.
2. Add `alb` module nodes and their configured ports as healthy backends in the external load balancer pool.
3. Shift traffic from Ingress NGINX Controller backends gradually.
4. Remove the old backends only after validation.

This supports node-by-node migration and weighted traffic distribution when the external load balancer provides those features.

When configuring backends, keep the following in mind:

- Configure health checks against `/healthz` on the `alb` module HTTP port.
- Preserve the original protocol and client-address handling: align Proxy Protocol settings for traffic and health checks, or configure trusted forwarded headers for an L7 load balancer.
- If both controllers run on the same nodes, use different host ports. Alternatively, select disjoint node sets.

Direct HostPort or HostWithFailover access

To switch traffic with direct HostPort or HostWithFailover access, do the following:

1. Deploy the `alb` module on a separate node set or use non-conflicting host ports.
2. For a node-by-node switch, add `alb` module node addresses to the DNS pool and remove Ingress NGINX Controller node addresses after each node passes validation.
3. Account for DNS TTLs and client-side caching: when records point to node addresses, DNS pools do not provide deterministic weighting or connection draining.

DNS cannot distinguish controllers that use different ports on the same node address. If clients must continue using ports `80` and `443`, use separate nodes or introduce a load balancer or NAT rule.

`HostWithFailover` has no direct equivalent in the `alb` module. When equivalent failover behavior is required, use the `LoadBalancer` inlet with MetalLB.

Validating the cutover

⚠ Traffic processing can differ between the Ingress NGINX Controller and the `alb` module, including generated and trusted headers, protocol handling, and supported features. Thoroughly test applications through the `alb` module path and, where necessary, adjust them to the traffic-processing behavior of the `alb` module before switching production traffic.

Before shifting production traffic:

- Confirm that the step 3 resources are applied.
- On the target ALBInstance or ClusterALBInstance status, check that `ready: true` and `synced: true`.
- Verify that `conflictPorts`, `conflictBackendTLS`, and `conflictFrontendTLS` are all `false`; if any is `true`, the matching `<CONFLICT>Owner` field names the controlling instance.
- Check the `Accepted` and `Programmed` conditions on the Gateway and ListenerSet status, and `Accepted` on the route status (`d8 k -n <NAMESPACE> get gateway,listenerset,httproute <NAME> -o yaml` — the `status.conditions` section).
- `True` in these conditions means the controller accepted the resource and applied its configuration.
- Test every hostname and protocol through the Gateway API entry-point address or with `migrationGateway`, including TLS certificates, redirects, authentication, long-lived connections, and application-specific policies.

- Verify preservation of the client address, Proxy Protocol or forwarded-header processing, source restrictions, and load balancer health checks.
- If HTTP-01 bridging is still required, keep `http01CertificateSolverBridging` enabled until DNS or the load balancer is switched.
- Confirm that metrics, logs, alerts, and dashboards identify traffic and errors on both controllers.
- Keep Ingress resources, the Ingress NGINX Controller, its external entry point, and valid certificates available throughout the rollback window.
- Choose the switching method above and define rollback criteria.

Rolling back

Define rollback criteria before the cutover. If validation fails:

1. For `migrationGateway` testing, remove `spec.migrationGateway`.
2. For a DNS switch, restore the Ingress NGINX Controller address and wait for the previous DNS TTL to expire.
3. For a manually managed load balancer, restore Ingress NGINX Controller nodes and ports in its backend pool.
4. If public traffic again enters through the Ingress NGINX Controller and the Issuer or ClusterIssuer resources for Gateway API certificates use HTTP-01 solvers, re-enable `http01CertificateSolverBridging`.

Do not delete the old Ingress resources, Ingress NGINX Controller, load balancer, or DNS values until rollback has been tested and the agreed stabilization period has elapsed.

Step 5. Cleanup

After the rollback window closes, do the following:

1. Remove `migrationGateway` if it is still set.
2. Disable `http01CertificateSolverBridging` and `ingress2Gateway` if they are still enabled.
3. Delete Ingress resources and Ingress NGINX Controllers that no longer serve applications or DP system interfaces.
4. Remove unused load balancers, certificates, Secret objects, and DNS records.
5. Restore normal DNS TTLs after confirming that no clients use the old entry point.

> **i** Some DP interfaces may still be published through the Ingress API. Do not disable the `ingress-nginx` module or delete related objects until the required interfaces are published through the Gateway API and validated. Follow “Publishing service domains”.

Additional resources

- ALB with Kubernetes Gateway API
- ALB with Ingress NGINX Controller
- Incoming traffic balancing overview
- Module `alb` documentation
- Module `alb` custom resources
- Module `alb` parameters
- Module `ingress-nginx` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

ALB with Istio

ALB with Istio is implemented via Istio Ingress Gateway or Ingress NGINX. The `istio` module is used for this purpose.

Use this option when you need advanced traffic management in a service mesh, for example canary routing or mTLS. Configuration details are in the `istio` module documentation.

Creating an IngressIstioController and preparing infrastructure is a cluster administrator task. Application publishing with Gateway and VirtualService resources, including canary deployment, is described in “Publishing applications using Istio Ingress Gateway resource”.

Ingress to publish applications

Applications can be published using the Istio Ingress Gateway or through Ingress NGINX with the Istio sidecar enabled.

Istio Ingress Gateway

To publish the application using the Istio Ingress Gateway, follow these steps:

1. Create an IngressIstioController resource.

   The example below creates a controller with the `HostPort` inlet on frontend nodes:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: IngressIstioController
   metadata:
     name: main
   spec:
     # The label selector value used to create the Gateway resource.
     ingressGatewayClass: istio-hp
     inlet: HostPort
     hostPort:
       httpPort: 80
       httpsPort: 443
     nodeSelector:
       node-role.deckhouse.io/frontend: ""
     tolerations:
       - effect: NoExecute
         key: dedicated.deckhouse.io
         operator: Equal
         value: frontend
     resourcesRequests:
       mode: VPA
   ```

2. Create a Secret with the TLS certificate and key for HTTPS. Place it in the `d8-ingress-istio` namespace and give application developers the Secret name and the `ingressGatewayClass` value:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: app-tls-secret
     namespace: d8-ingress-istio # Secret namespace is d8-ingress-istio, not the
   application namespace.
   type: kubernetes.io/tls
   data:
     tls.crt: |
       <TLS_CRT_DATA>
     tls.key: |
       <TLS_KEY_DATA>
   ```

Supported Secret formats are in the Istio guide on key formats.

Application developers then create the Gateway and VirtualService resources. Examples, including canary deployment, are in “Publishing applications using Istio Ingress Gateway resource”.

Ingress NGINX

To publish through Ingress NGINX with an Istio sidecar, enable the `enableIstioSidecar` parameter in the IngressNginxController of the `ingress-nginx` module and share the `ingressClass` name with application developers.

Ingress and Service manifests with the required annotations are in “Publishing applications using Ingress NGINX”.

Additional resources

- Publishing applications using Istio
- Incoming traffic balancing
- Module `istio` documentation
- Module `ingress-nginx` documentation
- Module `istio` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Ensuring high availability and fault tolerance (keepalived)

In Deckhouse Platform, the `keepalived` module can be used to provide high availability and fault tolerance.

To configure keepalived clusters, custom resources are used.

Module usage examples

Multiple public IP addresses

In the following example, there are three public IP addresses on three front nodes. Each virtual IP address is assigned to a separate VRRP group. Thus, each address “jumps” independently of the others, and if there are three nodes in the cluster with the `node-role.deckhouse.io/frontend: ""` labels, each IP address gets its own master node.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Required.
    node-role.deckhouse.io/frontend: ""
  tolerations:  # Optional.
  - key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
  vrrpInstances:
  - id: 1 # The unique cluster ID.
    interface:
      detectionStrategy: DefaultRoute # The card with the default route is used as a
service network one.
    virtualIPAddresses:
    - address: 42.43.44.101/32
      # The interface parameter is omitted since IP addresses are based on the cards
that service VRRP traffic.
  - id: 2
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.102/32
  - id: 3
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.103/32
```

In the following example, there is a gateway with a pair of IP addresses for LAN and WAN. In the case of the gateway, the private and public IPs are bind together, and they will “jump” between the nodes in tandem. In the below example, the VRRP traffic is routed through the LAN interface. It can be detected using the NetworkAddress method (assuming that each node has an IP address belonging to this subnet).

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
  - key: node-role.deckhouse.io/mygateway
    operator: Exists
  vrrpInstances:
  - id: 4 # Since "1", "2", "3" IDs are used in the "front" KeepalivedInstance above.
    interface:
      detectionStrategy: NetworkAddress
      networkAddress: 192.168.42.0/24
    virtualIPAddresses:
    - address: 192.168.42.1/24
      # In this case, we have already detected the local network (above); thus, the
interface parameter can be safely omitted.
    - address: 42.43.44.1/28
      interface:
        detectionStrategy: Name
        name: ens7 # The interface for public IPs is called "ens7" on all nodes,
therefore it needs to be named explicitly.
```

Switching keepalived manually

1. Enter the desired pod using a debug container with a shared process namespace:

   ```
   d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived
   ```

2. Edit the configuration file `vim /proc/1/root/etc/keepalived/keepalived.conf`, replace the value in the `priority` line with <number of keepalived pods + 1> or set a value higher than the current VRRP master (e.g., `255`).

3. Apply settings – send a signal to reload the configuration:

   ```
   kill -HUP 1
   ```

Additional resources

- Module `keepalived` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Egress Gateway configuration

Egress Gateway enables centralized management and control of outgoing traffic and provides features such as encryption, routing, and monitoring.

⚠ To use the Egress Gateway, enable the `cni-cilium` module in the cluster.

Egress Gateway usage modes

Basic mode

Pre-configured IP addresses are used on egress nodes.

Virtual IP mode

Egress Gateway lets you dynamically assign additional IP addresses to nodes.

Configuration principle

Configuring an egress gateway requires two custom resources:

1. EgressGateway: Describes the group of nodes, one of which will be selected as the active egress gateway, while the rest will remain in standby mode:
   - Among the group of nodes matching the `spec.nodeSelector` the eligible nodes will be selected. One of them will be assigned as the active gateway. The active node is selected in alphabetical order.

     Attributes of an eligible node:

     - The node is in `Ready` state.
       - The node is not in the maintenance state (not cordoned).
       - The `cilium-agent` on the node is in the `Ready` state.
     - When using EgressGateway in `VirtualIP` mode, an agent is launched on the active node which emulates a “virtual” IP address using the ARP protocol. The status of this agent’s pod is also taken into account when determining the eligibility of a node.
     - Different EgressGateways can use the same nodes for operation. The active node is selected independently for each EgressGateway, which allows for load balancing between them.
2. EgressGatewayPolicy: Describes the policy for routing network requests from pods in the cluster to a specific egress gateway defined using EgressGateway.

Node maintenance

To perform maintenance on a node that is currently the active egress gateway, follow these steps:

1. Remove the node label to exclude it from the egress gateway candidate pool. `<egress-label>` is the label specified in `spec.nodeSelector` of your EgressGateway.

   ```
   d8 k label node <node-name> <egress-label>-
   ```

2. Cordon the node to prevent new pods from starting:

   ```
   d8 k cordon <node-name>
   ```

   After this, Cilium will automatically select a new active node from the remaining candidates. Traffic will continue routing through the new gateway without interruption.

3. After maintenance is complete, return the node to service:

   ```
   d8 k uncordon <node-name>
   d8 k label node <node-name> <egress-label>=<value>
   ```

⚠ Reapplying the label may cause the node to become active egress gateway again (if it is first in alphabetical order among candidates).

To avoid immediate return to the active state, temporarily reduce the number of EgressGateway replicas or adjust selection priorities using additional labels.

Comparison with CiliumEgressGatewayPolicy

The `CiliumEgressGatewayPolicy` implies configuring only one node as an egress gateway. If it fails, there are no failover mechanisms and the network connection will be broken.

Configuration examples

EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # The "public" interface must have the same name on all nodes that matching the
nodeSelector.
     # If the active node fails, traffic will be redirected through the backup node
and
     # the source IP address of the network packets will change.
     interfaceName: eth1
```

EgressGateway in VirtualIPAddress mode

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # Each node must have all the necessary routes configured to access all
external public services,
     # the "public" interface must be prepared for automatic configuration of the
"virtual" IP as a secondary IP address.
     # In case of failure of the active node, traffic will be redirected through the
backup node and
     # the source IP address of the network packets will not change.
     ip: 172.18.18.242
     # List of network interfaces for Virtual IP.
     interfaces:
     - eth1
```

EgressGatewayPolicy

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
  - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
  - podSelector:
      matchLabels:
        app: backend
        io.kubernetes.pod.namespace: my-ns
```

Additional resources

- Module `cni-cilium` parameters
- Module `cni-cilium` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Inter-cluster cooperation (cluster alliance)

Deckhouse Platform (DP) supports two inter-cluster communication models:

- Multicluster
- Federation

Both models can be implemented using Istio (via the `istio` module). Istio provides deployment of a service mesh for managing and configuring network interactions between applications within a cluster.

Service mesh usage specifics in DP

Available scenarios depend on the platform edition.

Multicluster and federation are available in the Enterprise Edition (EE).

In the Community Edition (CE), Basic Edition (BE), Standard Edition (SE), and Standard Edition+ (SE+), the service mesh can only be used within a single cluster — for example, to implement patterns such as Circuit Breaker, Canary Deployment, and others.

Use case	CE / BE / SE / SE+	EE
Service mesh in one cluster	+	+
Federation	-	+
Multicluster	-	+

Differences between federation and multicluster

Federation connects independent (sovereign) clusters:

- Each cluster has its own namespace and services.
- Inter-cluster communication is configured explicitly. You define which services are accessible externally.

Multicluster tightly connects integrated clusters:

- A shared namespace is used.
- Services from one cluster are available to others as if they were local (unless restricted by authorization policies).

Models for combining multiple clusters (Istio example)

When using Istio, different combination models are possible:

- Network models: Clusters can be in the same or different networks.
- Control plane models: One or multiple service mesh control planes.
- Isolation models: Tenant isolation at the namespace, network, or cluster level.
- Service mesh models: One shared service mesh or multiple meshes.

These parameters can be combined to build an architecture tailored to the requirements of a specific infrastructure.

Problems solved with multicluster and federation

Combining clusters into a federation or multicluster can help solve the following tasks:

- Geographical distribution and GEO load balancing: Clusters serve users closer to their location.
- Fault tolerance: If a cluster or data center fails, traffic automatically switches to others (automatic failover).
- Horizontal scaling: Load distribution across clusters increases overall system performance.
- Environment and resource isolation: Different environments (production, staging, dev) can be placed in separate clusters, with optional interconnection.
- Regulatory compliance: For example, data storage requirements for specific geographic zones.
- Hybrid and multi-cloud infrastructure creation: Combining clusters located in different environments (cloud, different providers, etc.).

Additional resources

- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Multicluster

Multicluster of Istio funds

Requirements for clusters

- Cluster domains in the `clusterDomain` parameter of the ClusterConfiguration resource must be the same for all multicluster members. The default value is `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the ClusterConfiguration resource must be unique for each multicluster member.

  > - When analyzing HTTP and HTTPS traffic *(in Istio terminology)*, you can identify them and decide on further routing or blocking based on their headers.
  > - At the same time, when analyzing TCP traffic *(in Istio terminology)*, it is possible to identify them and decide on further routing or blocking based only on their destination IP address or port number.
  >
  > If the IP addresses of services or pods in different clusters match, requests from other pods in other clusters may mistakenly fall under the Istio's rules. The intersection of subnets of services and pods is not recommended (source).

By default, pods from different clusters communicate through the Istio ingress gateway (multi-network mode). If the pods of two clusters are directly reachable from each other (flat network), set `spec.enableIngressGateway` to `false` in both clusters, in the IstioMulticluster that describes the other cluster.

> ⚠ Only sidecar-mode workloads can take part in a multicluster. For details, refer to Ambient mesh limitations.

General principles

- Multicluster requires mutual trust between clusters to work. This is achieved by a mutual exchange of root certificates between clusters.
- Istio connects directly to the API server of the neighboring cluster to gather information about its services. The Deckhouse `istio` module establishes the secure communication channel.

Enabling the multicluster

Enabling the multicluster (via the `istio.multicluster.enabled = true` module parameter) results in the following:

- A proxy is added to the cluster to publish access to the API server via the standard Ingress resource:
  - Access through this public address is secured by authorization based on Bearer tokens signed with trusted keys.
  - Deckhouse automatically exchanges trusted public keys during the mutual configuration of the multicluster.
  - The proxy has read-only access to a limited set of resources.
- A service gets added to the cluster. That service exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - The public API server address (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).
  - Server public keys to authenticate requests to API server and to private metadata.

Managing the multicluster

To create a multicluster, you need to create a set of IstioMulticluster resources in each cluster that describe all the other clusters.

Example of configuring a multicluster of two clusters

Use the IstioMulticluster resource to configure a multicluster using Istio tools.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Additional resources

- Module `control-plane-manager` parameters
- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Federation

Federation with Istio (Service Mesh)

Requirements for clusters

- Each cluster must have a unique domain in the `clusterDomain` parameter of the `control-plane-manager` module. Note that none of the clusters should use the domain `cluster.local`, which is the default setting.

  > `cluster.local` can't be used as it's an unmodified alias for the local cluster domain. When specifying `cluster.local` as a principals in the AuthorizationPolicy, it will always refer to the local cluster, even if there is another cluster in the mesh with `clusterDomain` explicitly defined as `cluster.local` (for details, refer to the Istio documentation).

- There are no requirements for unique subnets of modules and services in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the `control-plane-manager` module when clusters are operating in a federation.

  > When analyzing traffic, Istio uses:
  >
  > - For HTTP and HTTPS requests — headers.
  > - For TCP requests — destination IP address and port number.
  >
  > Istio operates in the multi-network mode — pods from different clusters can only communicate with each other through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

> ⚠ Only sidecar-mode workloads can take part in a federation. For details, refer to Ambient mesh limitations.

General principles of federation

- Federation requires mutual trust between clusters. It requires a mutual root certificate exchange: cluster A must trust cluster B, and vice versa.
- Configuring inter-cluster access to services requires exchanging information about public services. In vanilla istio this is done using `ServiceEntry` resources. A `ServiceEntry` describes the public `ingressgateway` address of cluster B so that services in cluster A can reach the `bar` service in cluster B. The module creates ServiceEntries automatically (see below).

Enabling the federation — created services

Enabling federation (via the `istio.federation.enabled = true` module parameter) results in the following:

- The `ingressgateway` service is added to the cluster. Its task is to proxy mTLS traffic coming from outside to application services.
- Another added service exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - List of public services in the cluster (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).

Managing the federation

To establish a federation, you must:

- Create a set of IstioFederation resources in each cluster that describe all the other clusters.
  - After successful auto-negotiation between clusters, the IstioFederation resource will be filled with necessary service data in `status.metadataCache.public` and `status.metadataCache.private`.
- Add the `federation.istio.deckhouse.io/public-service: ""` label to each Service resource that is considered public within the federation.
  - In the other federation clusters, corresponding `ServiceEntry` and DestinationRule resources will be created for each such Service, leading to the `ingressgateway` of the original cluster.
  - The label value must be empty. You do not need to label other resources, such as Deployment, Pod, or VirtualService, to publish a service in the federation.

⚠ Federation publishing does not support `ExternalName` services, services without `.spec.ports`, or services with ports missing the `name` field.

Each port name must start with a supported Istio prefix: `http`, `http2`, `https`, `tcp`, `tls`, `grpc`, or `grpc-web`. The module uses the port name to determine the protocol in the generated `ServiceEntry`. If the prefix is not recognized, the port will be handled as TCP.

Example of a public service:

```
apiVersion: v1
kind: Service
metadata:
  name: reviews
  namespace: bookinfo
  labels:
    federation.istio.deckhouse.io/public-service: ""
spec:
  selector:
    app: reviews
  ports:
  - name: http
    port: 9080
    targetPort: 9080
```

To troubleshoot service publishing in the federation, follow these checks:

Check which services are marked as public in the local cluster:

```
d8 k get svc -A -l federation.istio.deckhouse.io/public-service=
```

Check metadata exchange status with remote clusters:

```
d8 k get istiofederation
d8 k get istiofederation <name> -o jsonpath='{.status.conditions}'
```

Check that the remote cluster provided its public services list:

```
d8 k get istiofederation <name> -o
jsonpath='{.status.metadataCache.private.publicServices}'
```

Check that local routing resources were created from the received metadata:

```
d8 k -n d8-istio get serviceentry,destinationrule
```

In the IstioFederation `status.conditions`, the `PublicMetadataExchangeReady`, `PrivateMetadataExchangeReady`, and `DataplaneConnectionReady` conditions should become `True`. If metadata exchange does not work, check the `D8IstioFederationMetadataEndpointDoesntWork` alert and the availability of the remote cluster `spec.metadataEndpoint`.

Example of configuring a federation of two clusters

To set up a federation with Istio, use the IstioFederation custom resource.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Additional resources

- Module `control-plane-manager` parameters
- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Network policies

Network policies restrict how pods communicate with each other, with external systems, and with cluster nodes. In Deckhouse Platform (DP), the implementation depends on the enabled CNI module.

Network policy implementation in DP

The available policy formats and the engine that processes them depend on the enabled CNI module:

- With the `cni-cilium` module, the implementation is built into Cilium and supports three policy formats:
  - the standard NetworkPolicy at L3 and L4;
  - CiliumNetworkPolicy — a namespaced resource with L3–L7 rules;
  - CiliumClusterwideNetworkPolicy — a cluster-scoped resource that also supports `nodeSelector` for protecting nodes (host firewall).
- With the `cni-flannel` module or another CNI without policy support, the `network-policy-engine` module handles enforcement on top of kube-router. Only the standard NetworkPolicy at L3 and L4 is supported. Policies are translated into `iptables` and `ipset` rules on every node.

> ⚠ Do not enable `cni-cilium` and `network-policy-engine` at the same time: Cilium already enforces network policies.

What is available in each engine

When choosing a policy format, consider what each engine supports:

- standard NetworkPolicy (L3/L4, namespaced) — supported by both engines;
- CiliumNetworkPolicy (L3–L7, FQDN, deny rules, namespaced) — only with `cni-cilium`;
- CiliumClusterwideNetworkPolicy (L3–L7, FQDN, deny rules, cluster-scoped) — only with `cni-`

`cilium`;
- node-level host firewall via CiliumClusterwideNetworkPolicy with `nodeSelector` — only with `cni-cilium`;
- policy audit mode (`policyAuditMode`) — only with `cni-cilium`.

Network infrastructure requirements

If the underlying infrastructure restricts network communication between servers, make sure the following conditions are met:

- Pod traffic tunneling is enabled: `tunnelMode` for CNI Cilium, `podNetworkMode` for CNI Flannel. Additionally, allow inter-node communication over the VXLAN port from the platform component network interaction list.
- Traffic between pod subnets (`podSubnetCIDR`) encapsulated in VXLAN is allowed, if the network inspects traffic inside the tunnel.
- Communication with external systems the cluster integrates with (LDAP, SMTP, external APIs) is allowed.
- Local communication inside each cluster node is allowed.
- Inter-node communication is allowed on the ports listed in the platform component network interaction list. Most ports are in the 4200–4299 range; new platform components are assigned ports from the same range when possible.

Sections

- Kubernetes NetworkPolicy — isolation model, selectors, default policies, API limitations.
- CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy — Cilium extensions, entities, L7 rules, FQDN rules, audit mode.
- Host firewall on nodes — protecting nodes with CiliumClusterwideNetworkPolicy and `nodeSelector`.
- Common policy examples — recipes for typical tasks.
- Diagnostics and observability — how to verify and debug policies.

Additional resources

- Network Policies — Kubernetes documentation
- Network Policy — Cilium documentation
- Overview of Network Policy — Cilium documentation
- Host Firewall — Cilium documentation
- Module `cni-cilium` documentation
- Module `network-policy-engine` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Kubernetes NetworkPolicy

The standard NetworkPolicy resource (`networking.k8s.io/v1`) defines L3/L4 traffic rules for pods (TCP, UDP, optionally SCTP). In DP, these policies are enforced by the `cni-cilium` module or the `network-policy-engine` module, depending on the CNI in use; the mapping between CNI and engine is described in Network policy implementation in DP.

Isolation model

NetworkPolicy describes what is allowed — there are no deny rules. By default, a pod is not isolated: all ingress and egress traffic is allowed. A pod becomes isolated as soon as any policy selects it via `spec.podSelector` and lists the matching direction in `spec.policyTypes`:

- A pod is isolated for ingress when a policy with `policyTypes: [Ingress]` selects it. Only the traffic listed in `ingress` is then allowed.
- A pod is isolated for egress when a policy with `policyTypes: [Egress]` selects it. Only the traffic listed in `egress` is then allowed.
- Reply traffic for allowed connections is always implicit.

Policies are additive: when several policies select the same pod, the resulting allow set is the union of all rules. Order of evaluation does not matter.

For a connection between a source pod and a destination pod, both the source's `egress` and the destination's `ingress` must allow it. If either side denies, the connection does not happen.

Selectors and fields

A policy uses the following selectors and fields:

- `spec.podSelector` — required. Defines which pods the policy applies to. The policy affects pods in the same namespace where it is created. An empty selector `{}` matches every pod in the namespace.
- `spec.policyTypes` — a list with `Ingress`, `Egress`, or both. If omitted, `Ingress` is always set; `Egress` is set only when the policy has egress rules.
- `ingress[].from` and `egress[].to` — sources and destinations (see below).
- `ingress[].ports` and `egress[].ports` — protocols and ports; a single port or a range via `endPort`.

Inside `from` and `to`, four selector types are available:

- `podSelector` — pods in the same namespace as the policy;
- `namespaceSelector` — all pods in the matching namespaces;
- `podSelector` and `namespaceSelector` in the same item — pods with the given labels in namespaces with the given labels;
- `ipBlock` — a CIDR range with an optional `except` list. Use it for cluster-external addresses, since pod IPs are ephemeral.

A complete example that uses every main construct:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-access
  namespace: my-app
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: frontend
        - podSelector:
            matchLabels:
              role: backend
      ports:
        - protocol: TCP
          port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
          port: 5978
```

This policy applies to pods labeled `app: db` in namespace `my-app`. Ingress on TCP/6379 is allowed from three sources: the CIDR `172.17.0.0/16` excluding `172.17.1.0/24`, any pod in namespace `frontend`, and pods labeled `role: backend` in the local namespace. Egress is allowed only on TCP/5978 to subnet `10.0.0.0/24`.

AND vs OR in selectors

A single array item with two selectors means AND (intersection):

```
ingress:
  - from:
      - namespaceSelector:
          matchLabels:
            user: alice
        podSelector:
          matchLabels:
            role: client
```

This allows traffic from pods labeled `role=client` in namespaces labeled `user=alice`.

Two separate array items mean OR (union):

```
ingress:
  - from:
      - namespaceSelector:
          matchLabels:
            user: alice
      - podSelector:
          matchLabels:
            role: client
```

This allows traffic from pods labeled `role=client` in the local namespace, or from any pod in namespaces labeled `user=alice`.

Selecting a namespace by name

The spec has no field that points to a namespace by name. Use the `kubernetes.io/metadata.name` label instead. Kubernetes automatically applies this label to every namespace, and its value equals the namespace name.

```
ingress:
  - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: frontend
```

Port ranges

The `endPort` field defines a port range:

```
egress:
  - to:
      - ipBlock:
          cidr: 10.0.0.0/24
    ports:
      - protocol: TCP
        port: 32000
        endPort: 32768
```

Constraints: `endPort` must be greater than or equal to `port`; `endPort` requires `port`; both values must be numeric.

Default policies for a namespace

Apply baseline policies to set the default behavior of a namespace:

- deny all ingress to the namespace’s pods:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
    - Ingress
```

- deny all egress from the namespace's pods:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
    - Egress
```

- deny both directions: list both values in `policyTypes` without rules.
- allow all ingress or egress: add an empty rule `ingress: [{}]` or `egress: [{}]`.

⚠ A default deny-egress policy blocks DNS too. If the pods use DNS, add a separate policy that allows egress to the kube-dns service in `kube-system` (UDP/53 and TCP/53).

Edge cases

`hostNetwork` pods

NetworkPolicy behavior for pods with `hostNetwork: true` is not defined by the API. Most engines, including Cilium and kube-router, treat such traffic as node traffic; `podSelector` and `namespaceSelector` do not match these pods. To filter such traffic, use `ipBlock` with the node IP or the host firewall on nodes.

Pod lifecycle

After a NetworkPolicy is created, the engine applies it asynchronously. A newly started pod selected by the policy may run for a short time without isolation rules or with partial rules. For critical dependencies, use init containers that wait for the required endpoints.

Existing connections

Behavior on policy changes during an open connection is implementation-defined: some engines tear down the connection, others let it finish. Avoid changing policies, pod labels, or namespace labels while important traffic is in flight.

L4 only

NetworkPolicy is defined for L4 (TCP, UDP, optionally SCTP). Behavior for other protocols (ICMP, ARP) depends on the engine and may differ.

Enforcement without Cilium: the `network-policy-engine` module

In clusters without Cilium, the `network-policy-engine` module enforces policies on top of kube-router. It deploys a DaemonSet in the `d8-system` namespace; kube-router runs as a Network Policy Controller and translates policies into `iptables` and `ipset` rules on every node:

- each policy becomes its own `KUBE-NWPLCY-*` chain;
- each isolated pod gets a `KUBE-POD-SPECIFIC-FW-*` chain;
- source and destination pod IPs are stored in ipsets, which keeps large rule sets compact and updates fast.

Only the standard Kubernetes formats are supported: `networking.k8s.io/NetworkPolicy API`, V1/GA, and beta semantics. Cilium extensions (CiliumNetworkPolicy, CiliumClusterwideNetworkPolicy,

L7 rules, FQDN, deny rules) are not supported by this engine.

Ready-to-use standard policy examples that work with both `network-policy-engine` and `cni-cilium` are available on the Common policy examples page.

API limitations

The Kubernetes NetworkPolicy API does not support the following scenarios (these are API-level limitations, not engine-level):

- L7 rules (HTTP, gRPC, Kafka, DNS-name filtering);
- deny rules — the model is “default deny + explicit allow”;
- selecting services by name;
- forcing all traffic through a shared gateway;
- targeting specific nodes by their Kubernetes identity (only via `ipBlock` with node IPs);
- logging — which connections were allowed or denied;
- blocking loopback or traffic from the pod’s own node.

Some of these tasks are addressed by CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy, available in clusters with `cni-cilium`.

Additional resources

- Network Policies — Kubernetes documentation
- Kube-router: Enforcing Kubernetes network policies with iptables and ipset
- Common policy examples
- Diagnostics and observability
- Module `cni-cilium` documentation
- Module `network-policy-engine` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy

In clusters with the `cni-cilium` module enabled, two Cilium-specific formats are available in addition to the standard NetworkPolicy:

- CiliumNetworkPolicy — a namespaced resource with L3–L7 rules;
- CiliumClusterwideNetworkPolicy — a cluster-scoped resource with the same rule language and `nodeSelector` support.

Cilium can enforce all three formats at the same time.

> ⚠ When NetworkPolicy, CiliumNetworkPolicy, and CiliumClusterwideNetworkPolicy are all in use, the resulting allow set is harder to reason about. Roll out new policies in audit mode and verify the behavior in Hubble.

How rules are evaluated

When combining rules from NetworkPolicy, CiliumNetworkPolicy, and CiliumClusterwideNetworkPolicy, Cilium follows these principles:

- deny rules take priority over allow rules;
- allow rules from NetworkPolicy, CiliumNetworkPolicy, and CiliumClusterwideNetworkPolicy are merged;
- if at least one policy applies to an endpoint (pod), the default-deny model takes effect for the corresponding traffic direction;
- L7 rules are evaluated only after L3/L4 checks pass.

What CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy add

Compared to the standard NetworkPolicy:

- L7 rules — HTTP, gRPC, Kafka, and DNS protocols;
- FQDN rules in egress — filtering by DNS names;
- deny rules — explicit denial of traffic;
- entities — sources and destinations of traffic such as `kube-apiserver`, `host`, `remote-node`, `world`;
- references to Kubernetes services by name or labels (`toServices`) — egress rules without specifying a CIDR;
- ICMP and ICMPv6 filtering by packet type;
- TLS filtering by Server Name Indication (SNI);
- `nodeSelector` (CiliumClusterwideNetworkPolicy only) — applies a rule to nodes themselves and is the basis for the host firewall on nodes;
- audit mode via `policyAuditMode` — log policy verdicts without enforcing them.

Resource structure

CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy share the same `spec` shape. A minimal example:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: example
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      app: db
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: frontend
      toPorts:
        - ports:
            - port: "5432"
              protocol: TCP
```

Key fields:

- `endpointSelector` — selects pods the policy applies to. Counterpart of `podSelector` in the standard NetworkPolicy.
- `nodeSelector` — selects nodes (CiliumClusterwideNetworkPolicy only). A single policy must use either `endpointSelector` or `nodeSelector`, not both.
- `ingress` and `egress` — rule arrays. Each rule has a peer field (`fromEndpoints`, `fromEntities`, `fromCIDR`, `fromCIDRSet`, `toEndpoints`, `toEntities`, `toCIDR`, `toCIDRSet`, `toFQDNs`, `toServices`) and an optional `toPorts` filter for protocols and ports.
- `ingressDeny` and `egressDeny` — deny rules. They are evaluated before allow rules.

Two special labels are useful in selectors. Cilium attaches them to every pod endpoint automatically:

- `io.kubernetes.pod.namespace` — the namespace where the pod runs. Use it in `fromEndpoints` and `toEndpoints` to reference pods in a specific namespace.
- `k8s-app`, `app`, and other regular pod labels — available without a prefix.

Egress to a Kubernetes service

The `toServices` field describes egress to a Kubernetes service rather than to a pod set. Match the service by name and namespace (`k8sService`) or by labels (`k8sServiceSelector`):

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-egress-to-redis
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: client
  egress:
    - toServices:
        - k8sService:
            serviceName: redis
            namespace: data
```

The policy automatically tracks changes to the service backends and applies the corresponding rules to them.

Entities

Entities are sources and destinations of traffic that make it easier to describe traffic to and from cluster components and infrastructure:

- `host` — the pod's own node, including the host's own traffic;
- `remote-node` — other cluster nodes;
- `kube-apiserver` — the Kubernetes API server (used by host firewall);
- `cluster` — every pod and node in the cluster;
- `world` — anything outside the cluster;
- `health` — Cilium health endpoints;
- `init` — containers that have not yet received an identity;
- `unmanaged` — pods not managed by Cilium;
- `all` — any entity.

An ingress rule that allows the API server to reach pods labeled `app: webhook`:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-apiserver-to-webhook
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      app: webhook
  ingress:
    - fromEntities:
        - kube-apiserver
      toPorts:
        - ports:
            - port: "9443"
              protocol: TCP
```

L4 rules: ICMP and SNI

In addition to ports and protocols, `toPorts` supports more L4 filters.

ICMP and ICMPv6

The `icmps` field allows or denies ICMP messages by packet type:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-icmp-echo
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: probe
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: monitoring
      icmps:
        - fields:
            - type: EchoRequest
              family: IPv4
```

Without an explicit `icmps` rule, ICMP traffic is blocked together with TCP and UDP once L4 filtering is active.

TLS Server Name Indication (SNI)

Egress can be restricted by SNI — the name a client sends in the TLS ClientHello. This filters access to external HTTPS services:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-egress-tls-sni
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: client
  egress:
    - toFQDNs:
        - matchPattern: "*.example.com"
      toPorts:
        - ports:
            - port: "443"
              protocol: TCP
          serverNames:
            - "api.example.com"
            - "static.example.com"
```

When `serverNames` is set, only TLS connections with the listed names are allowed; connections with any other SNI are blocked at the TLS handshake.

L7 rules

CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy can describe allowed application-level operations. L7 rules go inside `toPorts[].rules`:

> ⚠ L7 inspection is performed by the Envoy proxy embedded in the Cilium agent on each node. It adds per-connection latency and node CPU load. Avoid L7 rules on hot paths unless they are required — basic L3/L4 rules are sufficient for the common case.

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-readonly-api
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      app: api
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: client
      toPorts:
        - ports:
            - port: "8080"
              protocol: TCP
          rules:
            http:
              - method: GET
                path: "/api/v1/.*"
```

Clients labeled `app: client` may only call `GET /api/v1/...` on pods labeled `app: api`, port 8080.

Supported protocols: HTTP, gRPC, Kafka, DNS. Details and limitations are described in Layer 7 Examples in the Cilium docs.

Kafka

For Kafka, L7 rules allow specific operations (`apiKey`) and topics:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-kafka-produce
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: kafka
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: producer
      toPorts:
        - ports:
            - port: "9092"
              protocol: TCP
          rules:
            kafka:
              - role: produce
                topic: orders
```

Pods labeled `app: producer` may only publish (`role: produce`) to the `orders` topic. Any other operations, including `consume` and `metadata`, are denied at the Kafka protocol level.

DNS Policy and IP Discovery

When `toFQDNs` is in use, Cilium intercepts DNS responses allowed by `rules.dns` and updates an internal cache that maps DNS names to IP addresses. This cache is what `toFQDNs` rules check when allowing traffic. Therefore:

⚠ When `toFQDNs` is combined with DNS inspection (`rules.dns`), the application's DNS request goes through the Cilium proxy on the node — effectively doubling the resolve path. On high DNS traffic volumes this adds noticeable latency and `cilium-agent` load. Narrow `matchPattern` in `rules.dns` to the minimum required.

- DNS egress must be in the same policy as the FQDN rule, or in any other policy selecting the same pods;
- if a pod is not allowed to make DNS requests, its FQDN rules will not work;
- the TTL and lifetime of cache entries are determined by the Cilium agent based on DNS responses.

FQDN rules

Egress traffic can be restricted by DNS names via `toFQDNs`. For `toFQDNs` to work, DNS requests must be allowed and DNS inspection must be enabled through `rules.dns`. This can be done in the same policy or in any other policy that selects the same pods:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-egress-to-example
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      app: client
  egress:
    - toEntities:
        - cluster
      toPorts:
        - ports:
            - port: "53"
              protocol: UDP
            - port: "53"
              protocol: TCP
          rules:
            dns:
              - matchPattern: "*"
    - toFQDNs:
        - matchName: "api.example.com"
        - matchPattern: "*.cdn.example.com"
      toPorts:
        - ports:
            - port: "443"
              protocol: TCP
```

i The DNS egress rule uses `toEntities: cluster` rather than a label selector targeting `kube-dns` pods. DP deploys a `node-local-dns` DaemonSet alongside the main DNS service, so the actual DNS path from a pod may go through a `node-local-dns` instance. Using `toEntities: cluster` matches any cluster-internal DNS endpoint reliably.

Deny rules

Unlike the standard NetworkPolicy, CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy can explicitly deny traffic without removing broader allow rules:

```
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: deny-egress-to-metadata
spec:
  endpointSelector: {}
  egressDeny:
    - toCIDR:
        - 169.254.169.254/32
```

Deny rules are evaluated before allow rules and override permissions from any other policy.

Default policies via CiliumNetworkPolicy

As with the standard NetworkPolicy, a policy with empty rule lists places the selected endpoints into default-deny mode.

To put a namespace into default-deny mode, create a CiliumNetworkPolicy with empty rule lists:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: default-deny
  namespace: secure
spec:
  endpointSelector: {}
  ingress: []
  egress: []
```

DNS is also subject to network policies, so egress to the cluster DNS service over UDP/53 and TCP/53 must be allowed explicitly:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-dns
  namespace: secure
spec:
  endpointSelector: {}
  egress:
    - toEntities:
        - cluster
      toPorts:
        - ports:
            - port: "53"
              protocol: UDP
            - port: "53"
              protocol: TCP
```

i The DNS egress rule uses `toEntities: cluster` rather than a label selector targeting `kube-dns` pods. DP deploys a `node-local-dns` DaemonSet alongside the main DNS service, so the actual DNS path from a pod may go through a `node-local-dns` instance. Using `toEntities: cluster` matches any cluster-internal DNS endpoint reliably.

Audit mode (`policyAuditMode`)

The module parameter `policyAuditMode` puts Cilium into a mode where policies do not block traffic; they only log verdicts. This makes it safe to roll out large policy sets and verify them in Hubble UI before enforcement.

⚠ In audit mode, no network policy blocks traffic. Do not keep audit mode on permanently — turn it off after the rollout is done.

Recommended order:

1. Set `policyAuditMode: true` in the `cni-cilium` module configuration.
2. Apply the policy set. Apply node policies separately, following the procedure in Host firewall on nodes.
3. Inspect verdicts in Hubble UI and via `hubble observe --type policy-verdict`. Look for `verdict=AUDITED` entries — these connections would be blocked outside audit mode.
4. Adjust policies until the log only contains expected `verdict=ALLOWED` and `verdict=AUDITED` entries.
5. Turn audit mode off (`policyAuditMode: false`).

Once audit mode is off, policies start blocking traffic that is not allowed by any rule.

Additional resources

- Network Policy — Cilium documentation
- Overview of Network Policy — Cilium documentation
- Host firewall on nodes
- Common policy examples
- Diagnostics and observability
- Module `cni-cilium` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Host firewall on nodes

A host firewall is a Cilium mode where network policies apply to cluster nodes themselves rather than to pods. In DP, you configure it through CiliumClusterwideNetworkPolicy with the `nodeSelector` field. Available only in clusters with the `cni-cilium` module.

✖ A bug in host policies can break SSH access, control plane operation, kubelet, or etcd. Always roll out host firewall through `policyAuditMode` and verify verdicts in Hubble before enforcing.

How a host firewall differs from regular policies

A CiliumClusterwideNetworkPolicy with `nodeSelector` applies to a special host endpoint with the label `reserved:host` and filters traffic entering and leaving the node, including `hostNetwork` pods. Pod policies that use `endpointSelector` do not apply to the host endpoint — they target a different entity.

Host policies do not replace infrastructure-level protection (physical firewalls, cloud security groups). They are an additional filter layer inside the cluster.

Safe rollout

Roll out a host firewall in stages, using audit mode:

1. Set `policyAuditMode: true` in the `cni-cilium` module configuration. In audit mode, policies do not block traffic; they only log verdicts.
2. Apply the host policy set. At a minimum: a control plane policy (an example is provided below) and worker-node policies that allow SSH and required service ports.
3. Inspect verdicts in Hubble UI and via `hubble observe --type policy-verdict`. Expected traffic must be `verdict=ALLOWED`; anything in `verdict=AUDITED` would be blocked once audit mode is off.
4. Tune the policies until no unexpected `AUDITED` entries remain. Pay close attention to kubelet, etcd, kube-apiserver, ingress controllers, monitoring, and DNS.
5. Turn audit mode off (`policyAuditMode: false`).

If something breaks after enforcement, the fastest recovery path is to re-enable audit mode or delete

the CiliumClusterwideNetworkPolicy. Detailed recovery steps are in Cilium Emergency Recovery.

Mandatory rules

The host firewall policy set you maintain must explicitly configure the following permissions; otherwise, after audit mode is turned off, parts of the cluster will stop working:

- access from kube-apiserver to kubelet and webhook endpoints (ports 10250, 10255, and component webhook ports);
- inter-node access on etcd ports (2379, 2380), only between control plane nodes;
- access from worker nodes to the API server;
- BGP and service ports if MetalLB or a third-party load balancer is in use;
- platform component ports in the 4200–4299 range, listed in the platform component network interaction list;
- SSH from trusted administrative addresses;
- ICMP echo (optional, useful for diagnostics);
- DNS egress to kube-dns or an external resolver;
- access from monitoring to node-exporter and cilium-agent.

Tested policies for the most common cases are shown below: control plane connectivity, administrative SSH, and worker nodes. Use them as a starting set.

Use entities to describe peers:

- `host` — the node itself;
- `remote-node` — other cluster nodes;
- `kube-apiserver` — the Kubernetes API server;
- `cluster` — every pod and node in the cluster;
- `world` — anything external (combine with `toCIDR`/`fromCIDR` for narrowing).

Example: API server access for the control plane

A CiliumClusterwideNetworkPolicy that binds control plane nodes to the `kube-apiserver` entity and allows etcd traffic between control plane nodes. Without the `kube-apiserver` rule, control plane operations may stutter for up to a minute when `cilium-agent` pods restart, due to a Cilium Conntrack table reset:

```
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: allow-control-plane-connectivity
spec:
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/control-plane: ""
  ingress:
    - fromEntities:
        - kube-apiserver
    - fromEntities:
        - remote-node
      toPorts:
        - ports:
            - port: "2379"
              protocol: TCP
            - port: "2380"
              protocol: TCP
```

The `fromEntities: kube-apiserver` rule allows all traffic from the API server without port restrictions — this covers kubelet (TCP/10250, TCP/10255) and component webhook endpoints. The `fromEntities: remote-node` rule is scoped to etcd ports (TCP/2379, TCP/2380) and allows replication traffic between control plane nodes.

Example: SSH from administrative networks

Allows inbound SSH (TCP/22) on every node from the listed subnets only:

```
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: allow-ssh-admin
spec:
  nodeSelector: {}
  ingress:
    - fromCIDR:
        - 192.0.2.0/24
        - 198.51.100.10/32
      toPorts:
        - ports:
            - port: "22"
              protocol: TCP
```

Replace the subnets with the ones used for administrative access. Do not leave `fromEntities: [world]` without a narrowing CIDR — that is the same as an open SSH.

Example: baseline for worker nodes

Allows worker nodes to exchange traffic inside the cluster, reach the API server, and resolve DNS:

```
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: allow-worker-baseline
spec:
  nodeSelector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
  ingress:
    - fromEntities:
        - cluster
        - remote-node
  egress:
    - toEntities:
        - kube-apiserver
        - cluster
        - remote-node
```

This is a starting point. Extend the policy with rules for monitoring, ingress controllers, load balancers, and SSH that match your setup.

Additional resources

- Host Firewall — Cilium documentation
- CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy
- Platform component network interaction list
- Diagnostics and observability
- Module `cni-cilium` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Common policy examples

This page collects common scenarios for network policies. Standard NetworkPolicy examples work in

any cluster that supports network policies; CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy examples require the `cni-cilium` module.

The resource shape itself is documented in Kubernetes NetworkPolicy and CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy.

Deny all external ingress to a namespace, allow internal traffic

A baseline policy for a namespace where pods talk to each other but are not reachable from outside:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-external-ingress
  namespace: my-app
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector: {}
  egress:
    - {}
```

The empty `egress: [{}]` keeps all egress allowed; ingress is allowed only from pods in the same namespace.

Allow ingress from a specific namespace

Allows pods in the namespace labeled `kubernetes.io/metadata.name: frontend` to call pods labeled `app: api`:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-api
  namespace: my-app
spec:
  podSelector:
    matchLabels:
      app: api
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: frontend
      ports:
        - protocol: TCP
          port: 8080
```

Kubernetes sets the `kubernetes.io/metadata.name` label on every namespace automatically.

Allow egress to DNS and one CIDR only

Default-deny egress plus explicit DNS and one external service:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-egress
  namespace: my-app
spec:
  podSelector:
    matchLabels:
      app: client
  policyTypes:
    - Egress
  egress:
    - to:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system
          podSelector:
            matchLabels:
              k8s-app: kube-dns
      ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
          port: 53
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
          port: 5432
```

Allow specific pods to reach the API server

Through the `kube-apiserver` entity, Cilium tracks the API server IPs automatically — the rule does not need to be updated when addresses change:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: allow-controller-to-apiserver
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: controller
  egress:
    - toEntities:
        - kube-apiserver
      toPorts:
        - ports:
            - port: "443"
              protocol: TCP
```

Allow only GET requests to an API at L7

Clients labeled `app: client` may only call `GET /api/v1/...` on pods labeled `app: api`:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: readonly-api
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: api
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: client
      toPorts:
        - ports:
            - port: "8080"
              protocol: TCP
          rules:
            http:
              - method: GET
                path: "/api/v1/.*"
```

Allow egress to specific DNS names (FQDN)

For FQDN rules, Cilium must observe DNS queries, so the same policy must also allow egress to the cluster DNS with DNS inspection:

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: egress-to-fqdns
  namespace: my-app
spec:
  endpointSelector:
    matchLabels:
      app: client
  egress:
    - toEntities:
        - cluster
      toPorts:
        - ports:
            - port: "53"
              protocol: UDP
            - port: "53"
              protocol: TCP
          rules:
            dns:
              - matchPattern: "*"
    - toFQDNs:
        - matchName: "api.example.com"
        - matchPattern: "*.cdn.example.com"
      toPorts:
        - ports:
            - port: "443"
              protocol: TCP
```

i The DNS egress rule uses `toEntities: cluster` rather than a label selector targeting `kube-dns` pods. DP deploys a `node-local-dns` DaemonSet alongside the main DNS service, so the actual DNS path from a pod may go through a `node-local-dns` instance. Using `toEntities: cluster` matches any cluster-internal DNS endpoint reliably.

Deny access to the cloud metadata service

A cluster-scope deny rule blocks the cloud metadata service for every pod, even when other policies allow this egress:

```
apiVersion: cilium.io/v2
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: deny-egress-to-metadata
spec:
  endpointSelector: {}
  egressDeny:
    - toCIDR:
        - 169.254.169.254/32
```

Additional resources

- Kubernetes NetworkPolicy
- CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy
- Host firewall on nodes
- Diagnostics and observability
- Module `cni-cilium` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Diagnostics and observability

This page covers commands and techniques for inspecting applied network policies and investigating connectivity issues. Some tools require the `cni-cilium` module — this is noted explicitly.

Inspecting an applied policy

Resource summary:

```
d8 k describe networkpolicy <name> -n <namespace>
d8 k describe ciliumnetworkpolicy <name> -n <namespace>
d8 k describe ciliumclusterwidenetworkpolicy <name>
```

The output shows the selected pods or nodes, the resulting ingress and egress rules, and any validation errors.

List policies that affect a namespace:

```
d8 k get networkpolicy,ciliumnetworkpolicy -n <namespace>
d8 k get ciliumclusterwidenetworkpolicy
```

See which pods are isolated and which policies apply to them (Cilium clusters only):

```
d8 k -n d8-cni-cilium exec ds/agent -- cilium-dbg endpoint list
d8 k -n d8-cni-cilium exec ds/agent -- cilium-dbg endpoint get <endpoint-id>
```

In `cilium-dbg endpoint list`, each pod endpoint shows `POLICY (ingress)` and `POLICY (egress)` status: `Enabled`, `Disabled`, or `Disabled (Audit)` when `policyAuditMode` is on. Example output:

```
ENDPOINT   POLICY (ingress)   POLICY (egress)   IDENTITY   LABELS (source:key[=value])
1847       Enabled            Enabled            15234
k8s:app=client,k8s:io.kubernetes.pod.namespace=netpol-test
2156       Enabled            Disabled           28765
k8s:app=server,k8s:io.kubernetes.pod.namespace=netpol-test
3          Disabled           Disabled           1          reserved:host
```

`POLICY (ingress): Enabled` means at least one ingress policy applies to the pod and default-deny is active for inbound traffic. `POLICY (egress): Disabled` means there are no egress policies and outbound traffic is unrestricted.

Observability via Hubble

Hubble shows policy verdicts in real time and is the primary diagnostic tool in Cilium clusters.

In Hubble UI, connections between pods and services are tagged as `forwarded`, `dropped`, or `audit`. Drop events show which policy blocked the traffic and which rule field matched.

`hubble observe` can filter events by type. In DP, the `hubble` client ships with the agent, so it is convenient to run commands via `d8 k exec` against a cilium-agent pod:

```
d8 k -n d8-cni-cilium exec -it ds/agent -- hubble observe --type policy-verdict --verdict DROPPED
d8 k -n d8-cni-cilium exec -it ds/agent -- hubble observe --type policy-verdict --verdict AUDITED
d8 k -n d8-cni-cilium exec -it ds/agent -- hubble observe --from-pod my-app/client --to-pod my-
app/api
```

> **i** DP does not provide a standalone `d8 hubble` binary. Hubble CLI access is provided by running `exec` into a `cilium-agent` pod, as shown above. The `-it` flag is required for streaming output when `--last` is not specified.

Each agent sees events only for its own node. For cluster-wide event collection, use Hubble UI or export via HubbleMonitoringConfig.

Example output of `hubble observe --type policy-verdict --verdict DROPPED`:

```
Jun 10 12:00:01.234   netpol-test/outsider:52341   ->   netpol-test/server:8080   Policy verdict
INGRESS DENIED
Jun 10 12:00:01.236   netpol-test/outsider:52342   ->   netpol-test/server:8080   Policy verdict
INGRESS DENIED
```

Example output of `hubble observe --type policy-verdict --verdict AUDITED` (with `policyAuditMode` on):

```
Jun 10 12:05:01.101   netpol-test/client:53124   ->   netpol-test/server:8080   Policy verdict   INGRESS
AUDITED
```

The output includes policy and selector identifiers and the specific ingress/egress fields that matched, which makes it easy to find the rule that blocked or allowed the connection.

Continuous flow logs collection

For continuous flow log collection, enable export through the HubbleMonitoringConfig resource. Configuration is described in the cni-cilium examples.

Once export is on, `cilium-agent` writes events to `/var/log/cilium/hubble/flow.log` on every node. For central collection, use the `log-shipper` module with a ClusterLoggingConfig of type `File` that reads this file.

> **⚠** Updating HubbleMonitoringConfig restarts every Cilium agent in the cluster.

Diagnosing FQDN rules

If a `toFQDNs` rule does not allow traffic, inspect the DNS-name to IP cache maintained by `cilium-agent`:

```
d8 k -n d8-cni-cilium exec ds/agent -- cilium-dbg fqdn cache list
```

Example output when DNS requests have been intercepted:

```
Endpoint   Source Namespace   Source Name   FQDN           TTL    ExpirationTime              IPs
1847       netpol-test        client        example.com.   299    2026-06-10T12:05:00.000Z
93.184.216.34
```

The output shows entries with the source, DNS name, resolved IPs, and TTL. If there is no entry for the expected name, the pod either did not make a DNS request, or the DNS request is not allowed by a policy with DNS inspection (`rules.dns`). The cache mechanics are described in DNS Policy and IP Discovery.

Also check policy verdicts for DNS traffic:

```
d8 k -n d8-cni-cilium exec ds/agent -- hubble observe --type policy-verdict --port 53
```

Common mistakes

DNS broken after default-deny egress

A default-deny egress policy also blocks DNS. Add an egress rule to the kube-dns service in the `kube-system` namespace (UDP/53 and TCP/53). For details, refer to Default policies for a namespace.

AND vs OR mixed up

Two selectors in one `from`/`to` array item is AND; two separate items is OR. Refer to AND vs OR in selectors for the correct structure.

Policy does not apply to `hostNetwork` pods

Most engines, including Cilium and kube-router, treat such pods as node traffic. Use the host firewall on nodes to filter such traffic.

FQDN rule does not allow traffic

Cilium must observe DNS to keep the resolved IP set up to date. In any policy with `toFQDNs`, also allow egress to kube-dns and enable DNS inspection via `rules.dns`. For an example, refer to FQDN rules.

Connection drops after a policy change

Behavior for in-flight connections is not defined by the standard — some engines tear them down. Change policies during a maintenance window.

“Policy not applied” checklist

If a policy is created but traffic does not behave as expected, walk through these checks:

1. Which engine is enabled. The standard NetworkPolicy is supported by both engines; CiliumNetworkPolicy, CiliumClusterwideNetworkPolicy, L7, and FQDN require `cni-cilium`. The engine capabilities are listed in What is available in each engine.
2. The selector matches the pods: `d8 k get pods -n <namespace> -l <key>=<value>` should return the expected list.
3. `policyTypes` is correct. With `Ingress` only, egress stays unrestricted; with `Egress` only, ingress stays unrestricted.
4. AND vs OR in selectors. Re-check the array structure — a common cause of overly broad or overly narrow rules.
5. Audit mode. When `policyAuditMode` is on, policies do not block traffic. `cilium-dbg endpoint list` shows this as `Disabled (Audit)`.
6. Eventual consistency. Cilium and kube-router apply policies asynchronously. Wait a few seconds and re-test.
7. Policy status (CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy only). `d8 k get ciliumnetworkpolicy <name> -n <namespace> -o yaml` shows parse and apply errors in `status`.
8. Conflict with a deny rule. Cilium deny rules override any allow rules. Look for policies with `ingressDeny` or `egressDeny` selecting the same endpoint.

Additional resources

- Module `cni-cilium` custom resources
- Troubleshooting Policy — Cilium documentation
- Kubernetes NetworkPolicy
- CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy
- Host firewall on nodes
- Module `cni-cilium` documentation
- Module `log-shipper` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Software-defined networking (SDN)

Deckhouse Platform (DP) supports adding additional software-defined networks (SDN) to the cluster. The functions of additional software-defined networking (hereinafter referred to as additional networks) within DP are implemented using the `sdn` module.

Software-defined networking in DP allow declarative management of additional network segments for application workloads (pods, virtual machines). Instead of manually configuring network interfaces on each cluster node, the administrator describes the desired network state through custom Kubernetes resources, and the `sdn` module automatically configures the necessary network equipment.

DP supports the following features for working with software-defined networks:

- Configuration of network interfaces on nodes. Features such as port aggregation, bridging network interfaces, and configuring VLAN interfaces are supported.
- Additional networks for application workloads. Supports adding additional software-defined networks to the cluster: publicly available in each project (cluster) and available within a single namespace (project network).
- IPAM for additional networks. The mechanism allows you to allocate IP addresses (IPv4 addresses are supported) from pools and assign them to additional network interfaces of pods connected to cluster networks and project networks.
- Underlay networks for hardware device passthrough. This allows DPDK applications and other high-performance workloads to directly access physical network interfaces (PF/VF), bypassing the kernel network stack.
- System networks (service networks at nodes). These are designed for service traffic at the node level and are not used as additional pod networks. They are created on top of the existing underlay networks.

Additional resources

- Module `sdn` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring SDN in a cluster

To use SDN in a DP cluster, you need to prepare the infrastructure for enabling the `sdn` module, as well as perform some preparatory actions after enabling it.

Preparing the infrastructure for enabling the `sdn` module

Before using additional software-defined networks (hereinafter referred to as additional networks) in a cluster, preliminary infrastructure preparation is required:

- **For creating additional networks based on tagged VLANs:**
  - Allocate VLAN ID ranges on the data center switches and configure them on the corresponding switch interfaces.
  - Select physical interfaces on the nodes for subsequent configuration of tagged VLAN interfaces. You can reuse interfaces already used by the DP local network.
- **For creating additional networks based on direct, untagged access to a network interface:**
  - Reserve separate physical interfaces on the nodes and connect them into a single local network at the data center level.

Enabling the `sdn` module

Enable the `sdn` module according to the instructions.

Actions after enabling the sdn module

After enabling the module, NodeNetworkInterface objects will automatically appear in the cluster, reflecting the current state of the nodes.

To check for resources, use the command:

```
d8 k get nodenetworkinterface
NAME                           MANAGEDBY   NODE           TYPE     IFNAME          IFINDEX   STATE
AGE
virtlab-ap-0-nic-1c61b4a68c2a   Deckhouse   virtlab-ap-0   NIC      eth1            3         Up
35d
virtlab-ap-0-nic-fc34970f5d1f   Deckhouse   virtlab-ap-0   NIC      eth0            2         Up
35d
virtlab-ap-1-nic-1c61b4a6a0e7   Deckhouse   virtlab-ap-1   NIC      eth1            3         Up
35d
virtlab-ap-1-nic-fc34970f5c8e   Deckhouse   virtlab-ap-1   NIC      eth0            2         Up
35d
virtlab-ap-2-nic-1c61b4a6800c   Deckhouse   virtlab-ap-2   NIC      eth1            3         Up
35d
virtlab-ap-2-nic-fc34970e7ddb   Deckhouse   virtlab-ap-2   NIC      eth0            2         Up
35d
```

i When discovering node interfaces, the controller affixes the following labels, which are service labels (example):

```
labels:
  network.deckhouse.io/interface-mac-address: fa163eebea7b
  network.deckhouse.io/interface-type: NIC
  network.deckhouse.io/nic-pci-bus-info: 0000-17-00.0
  network.deckhouse.io/nic-pci-type: PF
  network.deckhouse.io/node-name: worker-01
annotations:
  network.deckhouse.io/heritage: NetworkController
```

In this example, each cluster node has two network interfaces: eth0 (DP local network) and eth1 (dedicated interface for additional networks).

Marking interfaces for organizing additional software-defined networks

To enable the configuration of additional software-defined networks, label the dedicated interfaces that are planned to be used for network creation (in the example above, eth1) with an appropriate label:

```
d8 k label nodenetworkinterface virtlab-ap-0-nic-1c61b4a68c2a nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-1-nic-1c61b4a6a0e7 nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-2-nic-1c61b4a6800c nic-group=extra
```

Combining multiple physical interfaces into a channel aggregation interface (bond interface)

To increase bandwidth or redundancy, it is possible to combine several physical interfaces into a bond interface (channel aggregation interface).

i Only network interfaces located on the same physical or virtual host can be combined.

Example configuring a bond interface:

1. Set custom labels on the interfaces intended for aggregation.

   Example of setting the `nni.example.com/bond-group=bond0` label on interfaces:

   ```
   d8 k label nni node-0-nic-fa163efbde48 nni.example.com/bond-group=bond0
   d8 k label nni node-0-nic-fa40asdxzx78 nni.example.com/bond-group=bond0
   ```

2. Prepare the configuration for creating the interface and apply it.

   Configuration example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterface
metadata:
  name: nni-worker-01-bond0
spec:
  nodeName: worker-01
  type: Bond
  heritage: Manual
  bond:
    bondName: bond0
    memberNetworkInterfaces:
      - labelSelector:
          matchLabels:
            # This is a service label that needs to be combined with the Bond
interface on a specific node.
            network.deckhouse.io/node-name: worker-01
            # Custom label (was added to the interfaces in the previous step).
            nni.example.com/bond-group: bond0
```

3. Check the status of the created Bond interface:

   Get a list of interfaces:

   ```
   d8 k get nni
   ```

   Example output:

   ```
   NAME                    MANAGEDBY   NODE                                    TYPE
   IFNAME      IFINDEX   STATE   AGE
   nni-worker-01-bond0     Manual      worker-01-b23d3a26-5fb4b-5s9fp      Bond
   bond0       76        Up      7m48s
   ...
   ```

   Check the status of the desired interface:

   ```
   d8 k get nni nni-worker-01-bond0 -o yaml
   ```

   Example of interface status:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterface
metadata:
...
status:
  conditions:
  - lastProbeTime: "2025-09-30T09:00:54Z"
    lastTransitionTime: "2025-09-30T09:00:39Z"
    message: Interface created
    reason: Created
    status: "True"
    type: Exists
  - lastProbeTime: "2025-09-30T09:00:54Z"
    lastTransitionTime: "2025-09-30T09:00:39Z"
    message: Interface is up and ready to send packets
    reason: Up
    status: "True"
    type: Operational
  deviceMAC: 6a:c7:ab:2a:a6:1e
  groupedLinks:
  - deviceMAC: fa:16:3e:92:14:40
    type: NIC
  ifIndex: 76
  ifName: bond0
  managedBy: Manual
  operationalState: Up
  permanentMAC: ""
```

Configuring and connecting additional virtual networks for use in application pods

The Deckhouse Platform provides the ability to declaratively manage additional networks for application workloads (pods, virtual machines). At the same time:

- Each additional network implies a single L2 data exchange domain.
- Within the Pod’s network namespace, an additional network is represented as a tap interface.
- The following modes are currently available for L2 network implementation:
  - **Tagged VLAN**: Communication between Pods on different Nodes uses VLAN-tagged packets and the infrastructure’s network equipment for switching. This method allows to create up to 4096 additional networks within a single cluster.
  - **Direct access to a Node’s network interface**: Communication between Pods on different Nodes uses the local network interfaces of the Nodes.
- From a network management perspective, there are two types of networks:
  - **Cluster network**: A network available in all projects, under administrator management. Example: a public WAN network or a shared network for cross-project traffic.
  - **Project network**: A network available within a Namespace, under user management.

Custom resources ClusterNetwork, Network, and NetworkClass are used to configure and connect additional networks for application pods.

> **i** If the VLAN type was specified in the Network or ClusterNetwork resources, NodeNetworkInterface will also be created for VLAN and Bridge.

> ⚠ Before creating an additional network, mark the interfaces that will be used by it.

Creating a publicly accessible network (cluster)

A custom resource ClusterNetwork is used to create publicly accessible networks across the entire cluster.

Creating a network based on tagged traffic

To create a network based on tagged traffic, follow these steps:

1. Create and apply the ClusterNetwork resource:

   In the `spec.type` parameter, specify the value `VLAN`. Tagged interfaces will be configured on the corresponding network interfaces of the nodes to ensure connectivity via the VLAN provided by the infrastructure.

   Example of a ClusterNetwork manifest for creating a public network based on tagged traffic:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterNetwork
   metadata:
     name: my-cluster-network
   spec:
     type: VLAN
     vlan:
       id: 900
     parentNodeNetworkInterfaces:
       labelSelector:
         matchLabels:
           # Manually applied label on NodeNetworkInterface resources.
           nic-group: extra
   ```

2. Check the status of the created resource with the command:

   ```
   d8 k get clusternetworks.network.deckhouse.io my-cluster-network -o yaml
   ```

   Example of the status of a ClusterNetwork resource:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterNetwork
   metadata:
   ...
   status:
     bridgeName: d8-br-900
     conditions:
     - lastTransitionTime: "2025-09-29T14:39:20Z"
       message: All node interface attachments are ready
       reason: AllNodeInterfaceAttachmentsAreReady
       status: "True"
       type: AllNodeAttachementsAreReady
     - lastTransitionTime: "2025-09-29T14:39:20Z"
       message: Network is operational
       reason: NetworkReady
       status: "True"
       type: Ready
     nodeAttachementsCount: 1
     observedGeneration: 1
     readyNodeAttachementsCount: 1
   ```

3. Check the connection of the additional network to the interfaces on the nodes .

Creating a network based on direct interface access

To create a network based on direct interface access, use the ClusterNetwork resource. In the `spec.type` parameter, specify the value `Access`. The corresponding network adapters on the nodes will be used directly to provide connectivity.

Example manifest for a network based on direct interface access:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: Access
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        # Manually applied label on NodeNetworkInterface resources.
        nic-group: extra
```

After creating the network, check the connection of the additional network to the interfaces on the nodes.

Creating a project network (user network)

In order for users to be able to create their own dedicated networks based on tagged traffic, it is necessary to first describe the range of tags available to them and define the network interfaces on which they can be configured. To do this, use the custom resource NetworkClass.

Example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NetworkClass
metadata:
  name: my-network-class
spec:
  vlan:
    idPool:
    - 600-800
    - 1200
    parentNodeNetworkInterfaces:
      labelSelector:
        matchLabels:
          nic-group: extra
```

Upon user request, the administrator provides them with the name of the created NetworkClass, which is used when creating the project network.

An example of creating a custom network using the NetworkClass resource administrator is described in the section “Creating a project network (user network)”.

Checking the connection of an additional network to interfaces on nodes

After creating ClusterNetwork or Network, the controller will create a NodeNetworkInterfaceAttachment tracking resource to link it to a NodeNetworkInterface.

To obtain a list of NodeNetworkInterfaceAttachment resources and information about a specific resource, use the following commands:

```
d8 k get nnia
d8 k get nnia my-cluster-network-... -o yaml
```

Example NodeNetworkInterfaceAttachment resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterfaceAttachment
metadata:
...
  finalizers:
    - network.deckhouse.io/nni-network-interface-attachment
    - network.deckhouse.io/pod-network-interface-attachment
  generation: 1
  name: my-cluster-network-...
...
spec:
  networkRef:
    kind: ClusterNetwork
    name: my-cluster-network
  parentNetworkInterfaceRef:
    name: right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b
  type: VLAN
status:
  bridgeNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-bridge-900
  conditions:
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Vlan created
      reason: VLANCreated
      status: "True"
      type: Exist
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Bridged successfully
      reason: VLANBridged
      status: "True"
      type: Ready
  nodeName: right-worker-b23d3a26-5fb4b-h2bkv
  vlanNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc
```

The NodeNetworkInterfaceAttachment status will change to `True` immediately after the corresponding NodeNetworkInterface appears and transitions to the `Up` state.

To check the status of NodeNetworkInterface, use the command:

```
d8 k get nni
```

Example output:

```
NAME                                                MANAGEDBY   NODE
TYPE     IFNAME      IFINDEX   STATE   AGE
...
right-worker-b23d3a26-5fb4b-h2bkv-bridge-900         Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   Bridge   d8-br-900   684       Up      14h
right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b   Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   NIC      ens3        2         Up      19d
right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc    Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   VLAN     ens3.900    683       Up      14h
...
```

IPAM: IP address pools for additional networks

The IPAM mechanism allows you to automatically allocate and assign **IPv4 addresses** for additional network interfaces of pods connected to cluster networks and project networks.

Principles and features of IPAM in DP

For each required IP address, a object IPAddress (ClusterIPAddress — for cluster networks) is created and used, which references the project network or cluster network. The controller allocates an address

from the pool and stores the result in `status.address`, `status.network`, `status.routes` of the IPAddress (ClusterIPAddress) object. The agent on the node assigns an IP address and routes to the interface inside the pod and sets the `status.conditions[Attached]` and `status.usedByPods` fields of the IPAddress (ClusterIPAddress) object.

Protection against conflicts when reusing IP addresses

To protect against conflicts, a cluster-scoped object IPAddressLease is created, which reserves the IP address. When an IPAddress object is deleted, the corresponding IPAddressLease is marked as `orphaned` (using the `status.orphaningTimestamp` field) and holds the address for the time specified in the `spec.ttl` parameter (to avoid rapid reuse).

Resources and settings for configuring IPAM

The following are used to manage the allocation and assignment of IP addresses:

- Address pools: for cluster networks (resource ClusterIPAddressPool) or project networks (resource IPAddressPool).
- Parameters for enabling IPAM for a specific network: `Network.spec.ipam.ipAddressPoolRef` — for project networks, `ClusterNetwork.spec.ipam.ipAddressPoolRef` — for cluster networks.
- The IPAddress resource is a request (automatic or static) for an address allocation, which is then assigned to the pod interface.

Example of assigning IP addresses to the cluster network

> **i** To allocate an address pool for a cluster network, use the ClusterIPAddressPool resource.

To allocate a pool of addresses and assign them to network interfaces of pods connected to the cluster network, perform the following steps:

1. Create an address pool. To do this, use the ClusterIPAddressPool resource.

   Example:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterIPAddressPool
   metadata:
     name: public-net-pool
   spec:
     leaseTTL: 24h
     pools:
       - network: 203.0.113.0/24
         ranges:
           - 203.0.113.10-203.0.113.200
   ```

   > **i** The `spec.pools[].ranges` parameter is optional. If it is not specified, the entire CIDR from `spec.pools[].network` is considered available (except for network/broadcast addresses, see the behavior of `/31` and `/32`).

2. Enable IPAM on the network. To do this, specify the parameters of the lusterIPAddressPool created in the previous step in the `spec.ipam.ipAddressPoolRef` parameter of the ClusterNetwork resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: VLAN
  vlan:
    id: 900
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        nic-group: extra
  ipam:
    ipAddressPoolRef:
      kind: ClusterIPAddressPool
      name: public-net-pool
```

After allocating a pool of IP addresses for the cluster network, they can be assigned to the network interfaces of pods connected to this network. For more details, see the section Assigning IP addresses to network interfaces of pods connected to an additional network.

Configuring and connecting underlay networks for hardware device passthrough

This feature is designed for high-performance workloads that require direct access to hardware, such as DPDK applications.

Key features

DP implements the following features for working with underlay networks:

- **Hardware device passthrough**: Physical network interfaces (PF/VF) are directly exposed to pods, bypassing the kernel network stack for maximum performance.
- **SR-IOV configuration**: Automatic configuration of SR-IOV on selected Physical Functions to create Virtual Functions, allowing multiple pods to share the same hardware.
- **DPDK support**: Devices can be bound in different modes suitable for DPDK workloads:
  - **VFIO-PCI**: Explicitly connects a network device to the pod by binding it to the `vfio-pci` driver. The corresponding VFIO device files (e.g., `/dev/vfio/vfio0`) are mounted into the pod for userspace access.
  - **DPDK**: A universal mode that automatically selects the appropriate driver for the network adapter vendor. For Mellanox NICs, the device is bound to the `mlx5_core` driver with both the netdev interface and necessary device files mounted (InfiniBand verbs files, `/dev/net/tun`, and the corresponding sysfs directory). For other vendors, the device is bound via VFIO (same as VFIO-PCI mode).
  - **NetDev**: Only the Linux network interface is passed through to the pod as a standard kernel network device.

Operation modes

The following device allocation modes are supported, which determine how physical interfaces are provided to hosts:

- **Shared mode**: Creates Virtual Functions (VF) from Physical Functions (PF) using SR-IOV, allowing multiple pods to share the same hardware. Each pod receives one or more VFs.
- **Dedicated mode**: Exposes each Physical Function as an exclusive device without SR-IOV. Each pod gets exclusive access to a complete PF, suitable for workloads requiring maximum performance.

Automatic interface grouping

When `autoBonding` is enabled, the controller groups interfaces from multiple matched PFs into a single DRA device. The interfaces are passed through to the pod as separate network interfaces, allowing applications (e.g., DPDK) to handle bonding/aggregation at the application level. Note that this does not create kernel-level bonding interfaces inside the pod.

Procedure for configuring and connecting physical interfaces to application subnets

To create underlay networks for forwarding hardware devices to pods, a custom resource UnderlayNetwork is used. It provides direct connection of physical network interfaces (Physical Functions and Virtual Functions) to pods via Kubernetes Dynamic Resource Allocation (DRA).

Prerequisites for DPDK applications

Before configuring UnderlayNetwork resources, you must prepare the cluster's worker nodes for DPDK applications:

- Configure hugepages.
- Configure Topology Manager.

Configuring hugepages

DPDK applications require hugepages for efficient memory management. Configure hugepages on all worker nodes using NodeGroupConfiguration:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: hugepages-for-dpdk
spec:
  bundles:
    - "*"  # Apply to all OS bundles.
  nodeGroups:
    - "*"  # Apply to all node groups.
  weight: 50  # Before the kubelet is configured and started.
  content: |
    #!/bin/bash
    echo "vm.nr_hugepages = 4096" > /etc/sysctl.d/99-hugepages.conf
    sysctl -p /etc/sysctl.d/99-hugepages.conf
```

This configuration sets `vm.nr_hugepages = 4096` on all nodes, providing 8 GiB of hugepages (4096 pages × 2 MiB per page).

Steps of NodeGroupConfiguration run in the order of `weight` together with the built-in steps, and the kubelet is configured at step 64. The weight is below 64, so hugepages are allocated before the kubelet is configured. The kubelet sees them on its first start, and the per-NUMA memory reservation accounts for them right away.

Configuring Topology Manager

Set the `SingleNumaNode` policy in `topologyManager` of the NodeGroup of worker nodes where DPDK applications will run. The kubelet then gives Guaranteed pods whole physical cores and allocates CPU, memory and devices of a pod from one NUMA node.

Example NodeGroup configuration:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  kubelet:
    topologyManager:
      policy: SingleNumaNode
    resourceReservation:
      mode: Auto
  nodeType: Static
```

A DPDK pod gets dedicated CPUs only if it is Guaranteed (requests equal limits) and requests a whole number of CPUs. With SMT enabled, the number of CPUs must be even. The whole pod must fit into one NUMA node.

For details, see How do I give pods dedicated CPUs and NUMA alignment?

Preliminary steps before creating an UnderlayNetwork

Before creating an UnderlayNetwork, ensure that:

1. Physical network interfaces (NICs) are available on the nodes and are discovered as NodeNetworkInterface resources.
2. The interfaces you plan to use are Physical Functions (PF), not Virtual Functions (VF).
3. For Shared mode, the NICs must support SR-IOV.

Checking and configuring network interfaces (NodeNetworkInterface)

First, check which Physical Functions are available on your nodes:

```
d8 k get nni -l network.deckhouse.io/nic-pci-type=PF
```

Example output:

```
NAME                            MANAGEDBY   NODE           TYPE   IFNAME   IFINDEX
STATE   VF/PF   Binding   Driver      Vendor   AGE
worker-01-nic-0000:17:00.0      Deckhouse   worker-01      NIC    ens3f0   3
Up      PF      NetDev    ixgbe       Intel    35d
worker-01-nic-0000:17:00.1      Deckhouse   worker-01      NIC    ens3f1   4
Up      PF      NetDev    ixgbe       Intel    35d
worker-02-nic-0000:17:00.0      Deckhouse   worker-02      NIC    ens3f0   3
Up      PF      NetDev    ixgbe       Intel    35d
worker-02-nic-0000:17:00.1      Deckhouse   worker-02      NIC    ens3f1   4
Up      PF      NetDev    ixgbe       Intel    35d
```

Label the interfaces that will be used for UnderlayNetwork:

```
d8 k label nni worker-01-nic-0000:17:00.0 nic-group=dpdk
d8 k label nni worker-01-nic-0000:17:00.1 nic-group=dpdk
d8 k label nni worker-02-nic-0000:17:00.0 nic-group=dpdk
d8 k label nni worker-02-nic-0000:17:00.1 nic-group=dpdk
```

i You can check the PCI information and SR-IOV support status for each interface:

```
d8 k get nni worker-01-nic-0000:17:00.0 -o json | jq '.status.nic.pci.pf'
```

The `status.nic.pci.pf.sriov.supported` section contains information about SR-IOV support.

Creating UnderlayNetwork in Dedicated mode

In Dedicated mode, each Physical Function is exposed as an exclusive device. This mode is suitable when:

- SR-IOV is not available or not needed.
- Each pod needs exclusive access to a complete PF.

To create an Underlay network in Dedicated mode, follow these steps:

1. Create and apply the UnderlayNetwork resource. In the `spec.mode` field, specify the value `Dedicated`.

   Example configuration:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-dedicated-network
spec:
  mode: Dedicated
  autoBonding: false
  memberNodeNetworkInterfaces:
    - labelSelector:
       matchLabels:
         nic-group: dpdk # Label used to mark interfaces during the verification
and configuration of network interfaces.
```

When `autoBonding` is set to `true`, all matched PFs on a node are grouped into a single DRA device, exposing all PFs to the pod as separate interfaces. When `false`, each PF is published as a separate DRA device.

2. Check the status of the created UnderlayNetwork:

   ```
   d8 k get underlaynetwork dpdk-dedicated-network -o yaml
   ```

   Example status of UnderlayNetwork in `Dedicated` mode:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: UnderlayNetwork
   metadata:
     name: dpdk-dedicated-network
   ...
   status:
     observedGeneration: 1
     conditions:
     - message: All 2 member node network interface selectors have matches
       observedGeneration: 1
       reason: AllInterfacesAvailable
       status: "True"
       type: InterfacesAvailable
   ```

Creating UnderlayNetwork in Shared mode

In `Shared` mode, Virtual Functions (VF) are created from Physical Functions (PF) using SR-IOV, allowing multiple pods to share the same hardware. This mode requires SR-IOV support on the NICs.

To create an Underlay network in `Shared` mode, follow these steps:

1. Create and apply the UnderlayNetwork resource. In the `spec.mode` field, specify the value `Shared`.

   Example configuration:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
spec:
  mode: Shared
  autoBonding: true
  memberNodeNetworkInterfaces:
    - labelSelector:
       matchLabels:
         nic-group: dpdk
  shared:
    sriov:
      enabled: true
      numVFs: 8
```

In this example:

- `mode: Shared` enables SR-IOV and VF creation.
- `autoBonding: true` groups one VF from each matched PF into a single DRA device.
- `shared.sriov.enabled: true` enables SR-IOV on selected PFs.
- `shared.sriov.numVFs: 8` creates 8 Virtual Functions per Physical Function.

> **i** The `mode` and `autoBonding` fields are immutable once set. Plan your configuration carefully before creating the resource.

2. After creating the UnderlayNetwork, monitor the SR-IOV configuration status:

```
d8 k get underlaynetwork dpdk-shared-network -o yaml
```

Example status of UnderlayNetwork in `Shared` mode:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
...
status:
  observedGeneration: 1
  sriov:
    supportedNICs: 4
    enabledNICs: 4
  conditions:
  - lastTransitionTime: "2025-01-15T10:30:00Z"
    message: SR-IOV configured on 4 NICs
    reason: SRIOVConfigured
    status: "True"
    type: SRIOVConfigured
  - lastTransitionTime: "2025-01-15T10:30:05Z"
    message: Interfaces are available for allocation
    reason: InterfacesAvailable
    status: "True"
    type: InterfacesAvailable
```

3. Verify that VFs have been created by checking NodeNetworkInterface resources:

```
d8 k get nni -l network.deckhouse.io/nic-pci-type=VF
```

Preparing namespaces for UnderlayNetwork usage

Before users can request UnderlayNetwork devices in their pods, the namespace must be labeled to enable UnderlayNetwork support. This is an administrative task that should be done for namespaces where DPDK applications will run:

```
d8 k label namespace mydpdk direct-nic-access.network.deckhouse.io/enabled=""
```

Configuring and connecting system networks (service networks)

System networks (service networks) are intended for internal traffic at the node level (e.g., for storage, management, etc.) and are not used as additional pod networks.

Additional service networks are created on cluster nodes on top of existing underlay networks. The custom resource SystemNetwork is used for this purpose. System networks obtain IP addresses from the ClusterIPAddressPool during IPAM configuration.

Principles and features of system networks:

- **Working on top of underlay networks**: A system network attaches to an underlay network (UnderlayNetwork). To connect, specify the name of the underlay network to which the system network should be connected in the `spec.underlayNetworkName` parameter of the SystemNetwork resource. The set of node interfaces (PF or VF) used by the system network is defined in the `memberNodeNetworkInterfaces` parameter of the UnderlayNetwork resource.
- **Support for different types of system network connections to the underlay network**: You can create a VLAN sub-interface (`type: VLAN`), use the underlay as-is (`type: Access`), or attach via an SR-IOV Virtual Function (`type: SRIOVVirtualFunction`) with optional tuning (MTU, MAC, spoof checking, etc.).
- **Support for IPAM mechanism**: Optional `spec.ipam` parameter references a ClusterIPAddressPool. The controller and agent then allocate addresses from that pool and assign them to the node interfaces for this system network.
- **Tracking the status of system networks**: The agent reports node addresses (including SystemNetwork IPs) in NodeNetworkStatus resources. Internal SystemNetworkNodeNetworkInterfaceAttachment resources track the binding of each system network to each node’s parent interface.

Prerequisites for creating and using system networks

To enable system networks in a cluster, the following requirements must be met:

1. An underlay network must exist. The system network will attach to it by name via `spec.underlayNetworkName` parameter. The underlay’s `memberNodeNetworkInterfaces` selectors determine which node interfaces are used.
2. **Optional**. To automatically assign IP addresses on interfaces belonging to a system network, create a system network address pool. It must be specified in the `spec.ipam.clusterIPAddressPoolName` parameter of the SystemNetwork resource when the network is created.

Creating a system network

To create a system network, use the SystemNetwork resource. Supported system networks have the following types of connections to the underlay networks on which they will operate:

- `VLAN`: VLAN interfaces are created on interfaces connected to the underlay network (underlay interfaces).
- `Access`: Direct access (without VLAN) to interfaces on nodes connected to the underlay network.
- `SRIOVVirtualFunction`: Connection to a physical interface via SR-IOV.

VLAN type

Creates a VLAN sub-interface on each matched underlay interface. When creating a network, you must specify the VLAN ID (parameter `spec.vlan.id`).

Example of a system network manifest with a `VLAN` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: storage-network
spec:
  type: VLAN
  underlayNetworkName: my-underlay
  vlan:
    id: 100
  ipam:
    clusterIPAddressPoolName: storage-pool
```

To check the network status after creation, use the section Checking system network status.

Access type

Uses the underlay interface as-is (no VLAN tag). Useful when the underlay already represents a single L2 segment.

Example of a system network manifest with a `Access` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: mgmt-network
spec:
  type: Access
  underlayNetworkName: my-underlay
  ipam:
    clusterIPAddressPoolName: mgmt-pool
```

To check the network status after creation, use the section Checking system network status.

SRIOVVirtualFunction type

Attaches via an SR-IOV Virtual Function. The underlay must be in `Shared` mode so that VFs exist. You can optionally tune VF settings.

Example of a system network manifest with a `SRIOVVirtualFunction` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: vf-service-network
spec:
  type: SRIOVVirtualFunction
  underlayNetworkName: dpdk-shared-network
  sriovVirtualFunction:
    vlan: 200
    mtu: 1500
    linkState: Auto
  ipam:
    clusterIPAddressPoolName: vf-pool
```

To check the network status after creation, use the section Checking system network status.

Creating a pool of IP addresses for configuring the IPAM system network

To create an address pool for configuring the system network IPAM, use the ClusterIPAddressPool resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterIPAddressPool
metadata:
  name: storage-pool
spec:
  leaseTTL: 24h
  pools:
    - network: 10.20.30.0/24
      ranges:
        - 10.20.30.10-10.20.30.250
```

When creating a system network, specify this pool in the `spec.ipam.clusterIPAddressPoolName` parameter of the SystemNetwork resource.

Checking system network status

To get a list of system networks, use the command:

```
d8 k get systemnetworks
```

To view the status of a specific system network, use the command:

```
d8 k get systemnetwork storage-network -o yaml
```

In `status` you will see:

- `nodeAttachementsCount`: Total attachments (one per matched node interface).
- `readyNodeAttachementsCount`: Attachments that are configured and ready.
- `conditions` — e.g. `Ready`: When all attachments are ready.

To view internal attachments (one per a “system network + parent interface” pair), use the command:

```
d8 k get systemnetworknodenetworkinterfaceattachments
```

To view IP addresses at the node level (including system network IPs) for all nodes, use the command (one NodeNetworkStatus is assigned to each node):

```
d8 k get nodenetworkstatus
```

To view information about IP addresses at the node level (including system network IPs) for a specific node, use the command:

```
d8 k get nodenetworkstatus -l network.deckhouse.io/node-name=worker-01 -o yaml
```

In `status.addresses` look for entries with `type: SystemNetworkIP` and `systemNetworkName` set to your system network name.

Additional resources

- Module `node-manager` documentation
- Module `sdn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual machine networking

The `.spec.settings.virtualMachineCIDRs` block lists the subnets in CIDR notation from which Deckhouse Platform (DP) assigns IP addresses to virtual machines, either automatically or on request.

Specify the subnet start address aligned to the mask, for example `192.168.1.192/27`, not an arbitrary address from the range.

Example:

```
spec:
  settings:
    virtualMachineCIDRs:
      - 10.66.10.0/24
      - 10.66.20.0/24
      - 10.77.20.0/16
```

The first and the last address of each subnet are reserved and never assigned to virtual machines. For example, in the `10.66.10.0/24` subnet, the `10.66.10.0` and `10.66.10.255` addresses are unavailable.

You can leave the block unset. The module still starts, but you can no longer work with virtual machine addresses:

- You can't create or use the VirtualMachineIPAddress resource.
- A virtual machine can't request the `Main` network in the `.spec.networks` parameter.
- The `.spec.networks` parameter of a virtual machine can't be empty.

⚠ The subnets in the `.spec.settings.virtualMachineCIDRs` block must not overlap with the cluster node subnets, the service subnet, or the pod subnet (`podCIDR`).

You can't delete a subnet if addresses from it are already assigned to virtual machines. You also can't clear the block once it's set.

Additional resources

- Module `virtualization` parameters
- Module `virtualization` custom resources

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing static routes and ip rules on cluster nodes

In Deckhouse Platform, static-routing and ip rule management on cluster nodes is implemented using the `static-routing-manager` module.

Configuration examples

Creating a route in main routing table

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-main
spec:
  ipRoutingTableID: 254 # Main routing table ID is 254.
  routes:
  - destination: 10.0.0.0/8
    gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
```

Based on this configuration, the route `10.0.0.0/8 via 192.168.0.1` will be created on the nodes

falling under `nodeSelector`:

```
ip -4 route ls
```

Example output:

```
10.0.0.0/8 via 192.168.0.1 dev eth0 realm 216
```

The `realm 216` instruction in the route is used as a marker to identify the route under module control (d8 hex = 216 dec).

Creating a route in additional routing table

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-extra
spec:
  routes:
  - destination: 0.0.0.0/0
    gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
status:
  ipRoutingTableID: 10000 # If spec.ipRoutingTableID is not specified, it will be
generated automatically and placed in status.
```

According to this configuration, the route `10.0.0.0.0/8 via 192.168.0.1` will be created on the nodes falling under `nodeSelector` in the table `10000`:

```
ip -4 route ls table 10000
```

Example output:

```
default via 192.168.0.1 dev eth0 realm 216
```

Creating an ip rule

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPRuleSet
metadata:
  name: myiprule
spec:
  rules:
  - selectors:
     from:
     - 192.168.111.0/24
     - 192.168.222.0/24
     to:
     - 8.8.8.8/32
     - 172.16.8.0/21
     sportRange:
       start: 100
       end: 200
     dportRange:
       start: 300
       end: 400
     ipProto: 6
    actions:
     lookup:
       routingTableName: myrt-extra
    priority: 50
  nodeSelector:
    node-role.deckhouse.io: load-balancer
```

According to this configuration, an ip rule will be created on the nodes falling under `nodeSelector`. To view results, run:

```
ip rule list
```

Example output:

```
...
50: from 192.168.111.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.222.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216
50: from 192.168.222.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.111.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216
...
```

Additional resources

- Module `static-routing-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing DNS in a Kubernetes cluster

DNS management in a Kubernetes cluster is implemented using the `kube-dns` module.

The module installs CoreDNS components for managing DNS in the Kubernetes cluster.

> The module deletes Deployments, ConfigMaps as well as RBAC for CoreDNS that were previously created using the `kubeadm` tool. When deploying your own CoreDNS, avoid using the names `coredns` or `system:coredns` for any resources (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Use alternative names like `infra-dns` to prevent automatic removal by Deckhouse Platform.

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
    - 8.8.8.8
    - 8.8.4.4
    hosts:
    - domain: one.example.com
      ip: 192.168.0.1
    - domain: two.another.example.com
      ip: 10.10.0.128
    stubZones:
    - zone: consul.local
      upstreamNameservers:
      - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
    - foo.bar
    - baz.qux
```

Additional resources

- Module `kube-dns` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Caching DNS requests on cluster nodes

In Deckhouse Platform, you can deploy a local caching DNS server on each cluster node. It exports metrics to Prometheus for visualization in a Grafana dashboard.

This feature is implemented by the `node-local-dns` module. The module consists of the original CoreDNS deployed as a DaemonSet on all cluster nodes, along with a network configuration algorithm and iptables rules.

Detailed information about the problems that caching DNS-server allows to solve and the principle of its work is available in Caching DNS server in a cluster.

Example custom DNS configuration in a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx
```

For details on DNS configuration, refer to the Kubernetes documentation.

Additional resources

- Module `node-local-dns` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

BpfLB operation modes for external traffic processing

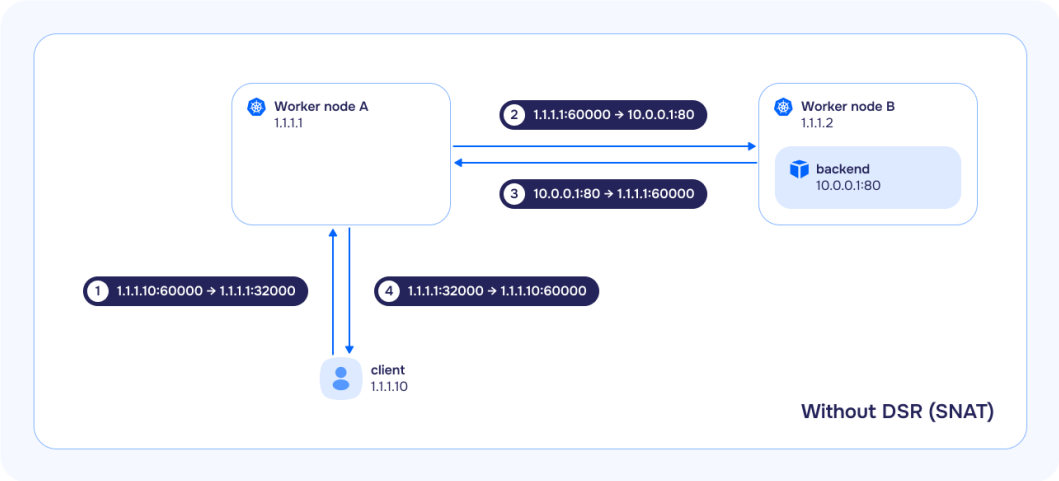
You can use the `cni-cilium` module to configure the BPF Load Balancer (bpfLB) mode of operation in Deckhouse Platform.

In Kubernetes, traffic typically comes through a load balancer that distributes incoming requests among servers and processes responses. Thus, the total throughput is limited by the load balancer resources.

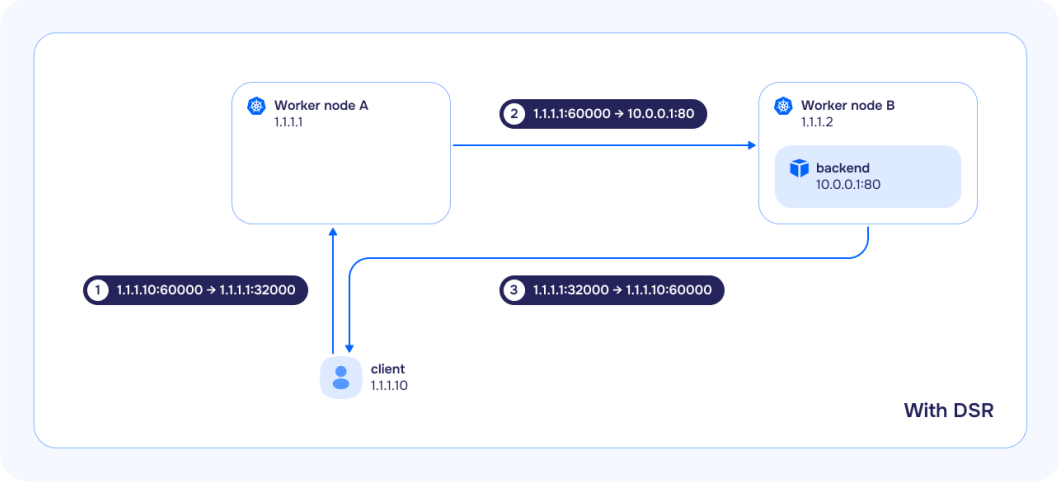
To optimize traffic and unload the balancer, the `DSR` mechanism is used, in which incoming packets go through the load balancer, and outgoing ones go directly from the terminating servers. Since responses are usually much larger in size than requests, this approach can significantly increase the overall throughput.

The module allows selecting the mode of operation via the `bpfLBMode` parameter, which affects the behavior of Services of the `NodePort` and `LoadBalancer` types:

- `SNAT` (Source Network Address Translation): A subtype of NAT in which, for each outgoing packet, the source IP address is translated to the IP address of the gateway from the target subnet, and incoming packets passing through the gateway are translated back based on a translation table. In this mode, `bpfLB` fully replicates the logic of `kube-proxy`:
  - If `externalTrafficPolicy: Local` is specified for a Service object, the traffic will be forwarded and balanced only to those target pods running on the same node where the traffic arrived. If the target pod is not running on this node, the traffic will be dropped.
  - If `externalTrafficPolicy: Cluster` is specified for a Service object, the traffic will be forwarded and balanced to all target pods in the cluster. At the same time, if the target pods are located on other nodes, SNAT will be performed when transmitting traffic to them (the source IP address will be replaced with the InternalIP of the node).



- `DSR` (Direct Server Return): A method where all incoming traffic passes through the load balancer, and all outgoing traffic bypasses it. This method is used instead of `SNAT`. Often, responses are much larger than requests, and `DSR` can significantly increase the overall throughput of the scheme:
  - If `externalTrafficPolicy: Local` is specified in the `Service`, its behavior is completely analogous to `kube-proxy` and `bpfLB` in `SNAT` mode.
  - If `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. Note the following:
    - If the target pods are on other nodes, then the source IP address will be preserved when incoming traffic is sent to them.
    - Outgoing traffic will go directly from the node on which the target pod was launched.
    - The source IP address will be replaced with the external IP address of the node to which the incoming request came originally.



> **i** In case of using `DSR` and `Service` mode with `externalTrafficPolicy: Cluster` additional network environment settings are required.
> Network equipment must be ready for asymmetric traffic flow: IP address anti-spoofing tools (`uRPF`, `sourceGuard`, etc.) must be disabled or configured accordingly.

- `Hybrid`: In this mode, TCP traffic is handled per `DSR` rules, while UDP traffic is handled per `SNAT` rules.

Example of bpfLB operation mode configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  settings:
    tunnelMode: VXLAN
    bpfLBMode: SNAT # SNAT mode is selected.
  version: 1
  enabled: true
```

Additional resources

- Module `cni-cilium` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Synchronizing time on nodes

To synchronize time on Deckhouse cluster nodes, DP uses a built-in solution based on chrony. Using the Network Time Protocol (NTP), DP ensures that system clocks on cluster nodes are synchronized with external NTP servers. If required, you can disable this built-in mechanism and configure custom NTP daemons.

Enabling built-in time synchronization

Enable the `chrony` module to activate time synchronization:

```
d8 system module enable chrony
```

By default, the time source is the server `pool.ntp.org`. You can specify a list of NTP servers using the `ntpServers` parameter in the configuration of the `chrony` module.

An example of the module configuration specifying NTP servers:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - ntp.ubuntu.com
      - time.google.com
  version: 1
```

Using custom NTP daemons

To disable the built-in time synchronization mechanism and use your own NTP daemons on the nodes, follow these steps:

1. Disable the `chrony` module:

   ```
   d8 system module disable chrony
   ```

   If the command is successful, you should see the message confirming that the module has been

disabled:

```
Module chrony disabled
```

2. Create a NodeGroupConfiguration resource to enable the NTP daemons on the nodes.

   Example for `systemd-timesyncd`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: enable-ntp-on-node.sh
spec:
  weight: 100
  nodeGroups: ["*"]
  bundles: ["*"]
  content: |
    systemctl enable systemd-timesyncd
    systemctl start systemd-timesyncd
EOF
```

Additional resources

- Module `chrony` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Access via OpenVPN

This section describes how to configure access to cluster resources using OpenVPN with certificate-based authentication.

Settings are available through a simple web interface that allows you to:

- Issue certificates;
- Revoke certificates;
- Undo certificate revocation;
- Download a ready-to-use user configuration file.

OpenVPN is integrated with user authentication, enabling access control for the web interface.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

External access to OpenVPN

You can configure the publication of the OpenVPN server in several ways:

1. Select one or more external IP addresses for connection.
2. Choose one of the connection methods:
   - via external IP address (`ExternalIP`) — if there are nodes with public IPs;

- using `LoadBalancer` — for all cloud providers and their deployment schemes that support LoadBalancer provisioning;
- `Direct` — manually configure the traffic path from the cluster entry point to the OpenVPN pod.

Example for bare-metal clusters

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: ExternalIP
    externalIP: 5.4.54.4 # External IP address.
```

Example for AWS and Google Cloud

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

Example for a public IP address on an external Load Balancer

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # Internal IP address that will receive traffic from the
external Load Balancer.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node
```

Authentication

You can configure user authentication to access OpenVPN. Alternatively, you can use the externalAuthentication parameter.
If both options are disabled, basic authentication with a generated password will be enabled.

To view the generated password, run the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values openvpn
-o json | jq '.openvpn.internal.auth.password'
```

To generate a new password, delete the Secret resource:

```
d8 k -n d8-openvpn delete secret/basic-auth
```

> **i** The `auth.password` parameter is no longer supported.

Available cluster resources after connecting to VPN

After connecting to the VPN, the following parameters are pushed to the user's computer:

- The `kube-dns` address is added to the client's DNS servers to allow direct access to Kubernetes services via FQDN;
- A route to the local network;
- A route to the cluster service network;
- A route to the pod network.

Auditing user connections

To monitor user connections, you can enable logging of VPN activity in JSON format. Traffic is grouped by the following fields: `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto`.
Using log-shipper, you can collect and store container logs for later auditing.

Why automatic DNS configuration doesn't work on macOS and Linux with the OpenVPN client

Due to the architectural specifics of Linux and macOS, automatic DNS server configuration when using the official OpenVPN client is not supported.

These operating systems allow using external scripts that run on VPN connect and disconnect to configure DNS.

In the client configurations generated by the module, the following blocks are predefined and commented out:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

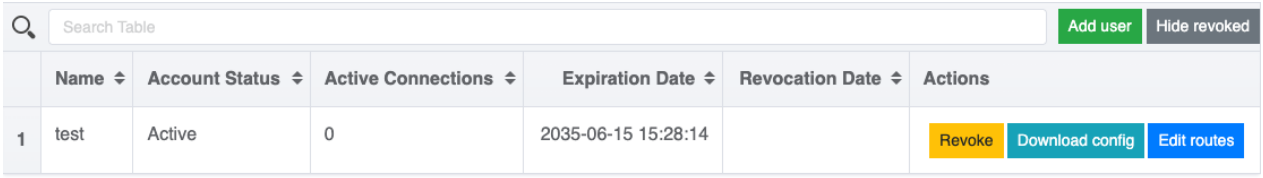
To activate these code blocks, you need to uncomment them (remove the leading `#`) and provide correct paths to the scripts.

You can either write your own scripts or use existing solutions provided by the official OpenVPN Community (for Linux).
For macOS, you can use a third-party script.

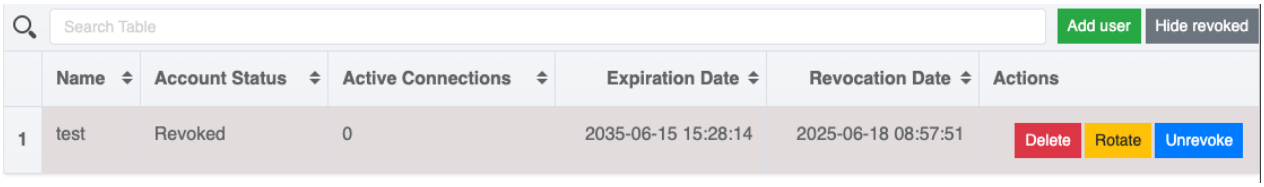
> **⚠** The scripts must have execution permissions.

How to revoke, rotate or delete a user certificate

All operations with client certificates are performed via the openvpn-admin web interface. Buttons for managing certificates are available to the right of each user's name:



To rotate (issue a new certificate) or delete a client, you must first revoke their current certificate:



Once revoked, the Renew (rotation) and Delete (removal) actions become available.

How to rotate a server certificate

The server certificate is rotated automatically several days before it expires.

If you need to rotate the certificate manually (e.g., due to certificate corruption or an unscheduled replacement), follow these steps:

1. Delete the secret `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-server
   ```

2. Restart the OpenVPN pods to trigger the generation of a new certificate:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

A new certificate will be generated automatically when the pods start.

How to rotate a root certificate (CA)

The root certificate (CA) and server certificate are rotated automatically 1 day before expiration. Automatic rotation of clients certificates is not provided. The root certificate (CA) is used to sign all certificates in OpenVPN — both server and client. Therefore, when replacing the CA, you must reissue all dependent certificates.

Steps to rotate the root certificate:

1. Revoke or delete all active client certificates using the openvpn-admin web interface. If you choose to revoke certificates, you can rotate them later (Renew) after the CA is replaced, without recreating the clients.
2. Delete secrets `openvpn-pki-ca` and `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
   ```

3. Restart OpenVPN pods:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

4. Rotate certificates of revoked users, or create new ones.
5. Delete all revoked certificate secrets:

   ```
   d8 k -n d8-openvpn delete secrets -l revokedForever=true
   ```

Additional resources

- Module `log-shipper` documentation
- Module `openvpn` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage

Reliable data storage is one of the key tasks when deploying and operating Kubernetes clusters. In Deckhouse Platform, this is achieved through flexible support for both software-defined and external storage systems, as well as convenient automation and management tools.

In this section, you will learn:

- Which types of storage Deckhouse supports.
- How to set up both local (SDS) and external storage systems.
- Which tools Deckhouse provides to simplify storage configuration and operation.
- How to choose the optimal solution for your scenarios, based on requirements for reliability, performance, and scalability.

For more details on disk layout and recommended sizes, see the “Disk layout and size” guide.

Supported storage types

Deckhouse Platform offers a wide range of solutions, which can be divided into two main groups.

Software-defined storage

- Managing logical volumes on cluster nodes
- Local storage using Local Path Provisioner
- Local storage based on LVM (Logical Volume Manager)
- Replicated storage based on DRBD (Distributed Replicated Block Device)

External storage

- Distributed Ceph storage
- HPE data storage
- Huawei data storage
- NFS data storage
- S3-based object storage
- SCSI-based data storage
- TATLIN.UNIFIED (Yadro) unified storage

Key features

- All storage configurations are performed through Deckhouse and its modules, simplifying integration with all cluster components.
- Ready-to-use solutions for Ceph, NFS, S3, corporate storage systems, and other options, which make connecting storage to the cluster straightforward.
- Thanks to SDS support (including DRBD) and distributed solutions (Ceph, S3), you can scale up as your workload grows.
- Data replication and integration with robust storage systems help protect critical services and applications from data loss.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage configuration

Storage setup involves several steps depending on the selected storage type. The main configuration steps include:

- Enabling and configuring the corresponding modules.
- Creating Volume Groups.
- Preparing and creating StorageClass objects, followed by assigning and using them.

Each storage type may have its own specific requirements and configuration details, which are described in the corresponding guides.

Creating a StorageClass

To create StorageClass objects, you need to connect one or more storage backends that manage PersistentVolume resources. The created StorageClass objects can then be used to provision virtual disks and images. For more information on creating and using StorageClasses, refer to the appropriate documentation sections for each storage type.

Setting a default StorageClass

The default StorageClass is used when a PersistentVolumeClaim is created without explicitly specifying a storage class. This simplifies the process of creating and using storage by eliminating the need to define the class manually each time.

To set the default StorageClass, specify the desired class in the global configuration. Example command:

```
# Replace with the name of your StorageClass object.
DEFAULT_STORAGE_CLASS=replicated-storage-class
d8 k patch mc global --type='json' -p='[{"op": "replace", "path":
"/spec/settings/defaultClusterStorageClass", "value":
"'"$DEFAULT_STORAGE_CLASS"'"}]'
```

Additional resources

- Global parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring volume snapshot creation

Deckhouse Platform supports volume snapshot creation for CSI drivers in a Kubernetes cluster.

Snapshots capture the state of a volume at a specific point in time and can be used for data recovery or volume cloning. The ability to create snapshots depends on the capabilities of the CSI driver in use.

Supported CSI drivers

The following CSI drivers support snapshot creation:

- OpenStack cloud provider
- VMware vSphere cloud provider
- Distributed storage based on Ceph
- Amazon Web Services cloud provider
- Microsoft Azure cloud provider
- Google Cloud Platform cloud provider
- Replicated storage based on DRBD
- NFS-based storage

Creating snapshots

Before creating snapshots, make sure that VolumeSnapshotClass resources are configured in the cluster. You can list available classes with the following command:

```
d8 k get volumesnapshotclasses.snapshot.storage.k8s.io
```

To create a snapshot for a volume, specify the required VolumeSnapshotClass in the manifest:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-snapshot
spec:
  volumeSnapshotClassName: <class-name>
  source:
    persistentVolumeClaimName: <pvc-name>
```

Restoring from a snapshot

To restore data from a snapshot, create a PVC that references the existing VolumeSnapshot:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restored-pvc
spec:
  dataSource:
    name: example-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  storageClassName: <storage-class-name>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

⚠ Not all CSI drivers support restoring volumes from snapshots. Ensure that your driver provides the required capabilities.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `cloud-provider-azure` documentation
- Module `cloud-provider-gcp` documentation
- Module `cloud-provider-openstack` documentation
- Module `cloud-provider-vsphere` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Local path provisioner storage

Deckhouse Platform provides the ability to configure local storage using Local Path Provisioner. This is a simple solution without support for snapshots or volume size limits, best suited for development, testing, and small clusters. It uses local disk space on Kubernetes nodes to create PersistentVolumes without relying on external storage systems.

How it works

For each LocalPathProvisioner resource, a corresponding `StorageClass` object is created. The list of nodes allowed to use the StorageClass is defined by the `nodeGroups` field and is used when scheduling pods.

When a pod requests a disk, the following occurs:

- A PersistentVolume of type `HostPath` is created;
- A directory is created on the appropriate node, with the `path` generated from the path parameter, the PV name, and the PVC name.

Example path:

```
/opt/local-path-provisioner/pvc-d9bd3878-f710-417b-a4b3-38811aa8aac1_d8-monitoring_prometheus-main-db-prometheus-main-0
```

Limitations

- It is not possible to set a size limit for created volumes.

Example LocalPathProvisioner resources

ReclaimPolicy: Retain (default)

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

ReclaimPolicy: Delete

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
  reclaimPolicy: "Delete"
```

Configuring Prometheus to use local storage

1. Apply the LocalPathProvisioner resource:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: LocalPathProvisioner
   metadata:
     name: localpath-system
   spec:
     nodeGroups:
     - system
     path: "/opt/local-path-provisioner"
   ```

2. Ensure that `spec.nodeGroups` matches the NodeGroup where Prometheus will be running.
3. Specify the name of the created StorageClass in the Prometheus configuration:

   ```
   longtermStorageClass: localpath-system
   storageClass: localpath-system
   ```

4. Wait for Prometheus pods to restart.

Additional resources

- Module `local-path-provisioner` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Node configuration for software-defined storage

Deckhouse Platform provides automatic management of logical volumes (Logical Volume Manager, LVM) on cluster nodes using custom Kubernetes resources. This functionality is provided by the `sds-node-configurator` module and includes:

- Detection of block devices on each node and creation of corresponding BlockDevice resources.

  > Manual creation and modification of the BlockDevice resource is prohibited.

- Automatic discovery and management of LVMVolumeGroup resources with the label `storage.deckhouse.io/enabled=true` (including thin pools) on cluster nodes. When Volume Groups (VG) are detected on nodes without corresponding resources, the controller automatically creates the LVMVolumeGroup resource.
- Regular scanning of Physical Volumes (PV) that are part of managed Volume Groups (VG) on nodes. Upon expansion of the underlying block device, the controller performs `pvresize` on the corresponding physical volume and automatically increases the size of all logical volumes in this volume group.

  > Reducing the size of the block device is not supported.

- Synchronization of the LVMVolumeGroup state with the actual state of the node: creation of new volume groups according to LVMVolumeGroup, expansion of existing ones when `desiredCapacity` changes, and deletion of the group when the resource is deleted. For detailed examples of operation, see the section Examples of working with LVMVolumeGroup.

Additional resources

- Module `sds-node-configurator` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

The controller operates with two types of Kubernetes custom resources:

- BlockDevice: A resource representing a block device.
- LVMVolumeGroup: A resource describing a Logical Volume Manager volume group.

Working with BlockDevice resources

Creating a BlockDevice resource

The controller periodically scans available block devices on each node and automatically creates a BlockDevice resource when it detects a device that matches the rules. As a result, a resource object with a unique name is created, containing detailed information about the device's characteristics.

Criteria for device selection by the controller

- The device is not used as a DRBD volume (Distributed Replicated Block Device).
- The device is not a loop interface (virtual block device).
- The device is a logical volume.
- The file system is absent or has the label `LVM2_MEMBER` (indicating it belongs to an LVM device).
- The device does not contain partitions (no partition table).
- The device's capacity exceeds 1 GiB.
- For virtual disks, a serial number is required.

The created BlockDevice resource serves as a data source for subsequent work with LVMVolumeGroup resources.

Updating a BlockDevice resource

When the state of a block device changes (for example, size change, metadata change, or the device disappears from the system), the controller automatically detects these changes and updates the fields of the BlockDevice resource, ensuring the information is up-to-date. User edits to the BlockDevice resource are prohibited and will be overwritten by the controller.

Deleting a BlockDevice resource

The controller deletes a BlockDevice resource only when the following conditions are met:

- The resource is in the `Consumable` state (available for consumption in the Logical Volume Manager).
- The block device is no longer available in the system (the device has been removed or disconnected).
- The block device is not part of a volume group with the label `storage.deckhouse.io/enabled=true` (such a volume group is managed by LVMVolumeGroup, and the BlockDevice is not deleted).

> **i** A BlockDevice resource is deleted without a help from a user. If the user manually deletes the resource, the controller will recreate it during the next scan if the device still meets the criteria.

Working with LVMVolumeGroup resources

The LVMVolumeGroup resource combines several BlockDevice resources into a single Logical Volume Manager (LVM) volume group on a node and reflects the current state of this group.

> **i** Currently, only the `Local` type (local volume groups) is supported.

Creating an LVMVolumeGroup resource

There are two scenarios for creating an LVMVolumeGroup resource:

- Automatic creation of the resource:
  - The controller scans the list of active LVM volume groups on each node.
  - If a discovered volume group has the label `storage.deckhouse.io/enabled=true` and there is no corresponding LVMVolumeGroup resource, the controller automatically creates the resource.
  - The controller fills out the `spec` section, except for the `thinPools` field. All other parameters (block device names, Volume Group name, etc.) are automatically pulled from the system state.
  - To manage thin pools, the user can manually add information about them to the `spec` section after the resource is created.
- Manual creation of the resource:
  - The user creates a YAML manifest, specifying the minimum set of fields:
    - `metadata.name`: Unique name for the LVMVolumeGroup resource.
    - `spec`: Name of the node where the volume group will exist.
  - After validation, the controller creates the volume group on the node and updates the custom

resource with current information about the state of the created LVM Volume Group.

Example of creating a local Volume Group without a thin pool:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd644126□7
  actualVGNameOnTheNode: "vg-0"
```

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"
```

Example of creating a local Volume Group with a thin pool (250 GiB):

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd644126□7
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi
```

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi
```

The user can use any valid selectors (`matchLabels` or `matchExpressions`) to select BlockDevice resources:

- `matchLabels` allows selecting all devices that have the specified label (for example, `kubernetes.io/hostname=node-0`).
- `matchExpressions` allows for more flexible expressions, such as including listed device names.

The `spec.local.nodeName` field must match the name of the node where the Volume Group is created. Otherwise, the resource will not be started. All selected block devices must physically reside on the same node for a resource of type Local.

⚠ After applying the manifest, the controller will automatically create or update the LVM infrastructure on the node, including working with physical volumes (`pvcreate`), creating or modifying the volume group (`vgcreate`/`vgextend`), and setting up the thin pool (`lvcreate`).

Updating an LVMVolumeGroup resource

To change the configuration of the volume group (adding/removing block devices or changing thin pool parameters), it is sufficient to edit the `spec` section of the resource. The controller will validate the new parameters and adjust the state on the node (execute commands like `vgextend`, `lvcreate`, `lvresize`, etc.).

The `status` section of the resource automatically reflects the current information about the state of the Volume Group (list of physical volumes, thin pool data, free/used space, etc.).

⚠
- The controller does not change the `spec` automatically — it only reads the desired state and brings the actual state of the node into compliance. All changes to the desired state must be made by the user in `spec`.
- Avoid modifying `status` manually, as this section is managed by the controller.

Deleting an LVMVolumeGroup resource

The controller automatically deletes the resource if the Volume Group no longer exists on the node (for example, all block devices are disconnected or removed). For manual deletion of the Volume Group and its associated logical volumes, the user can use:

```
d8 k delete lvg <resource-name>
```

After that, the controller will perform a full removal of the physical and logical LVM objects (`lvremove`, `vgremove`, etc.).

⚠ If there are logical volumes (including thin pools) in the Volume Group, the user must first delete or move them (using commands `lvremove` or `pvmove` + `vgreduce`). Otherwise, the controller will not be able to delete the resource. To prevent accidental deletion of the resource, you can set the annotation `storage.deckhouse.io/deletion-protection` — while it is present, the controller will not initiate the deletion of the volume group.

Removing a BlockDevice from an LVMVolumeGroup

1. To exclude a specific BlockDevice from the Volume Group, adjust the `spec.blockDeviceSelector` selector of the LVMVolumeGroup resource or remove the corresponding label from the BlockDevice resource.
2. On the node, manually transfer data (if there are volumes in the Volume Group) using the following command:

   ```
   pvmove <source-device>
   ```

3. Remove the physical volume from the group:

   ```
   vgreduce <VG-name> <excluded-device>
   ```

4. Finally, remove the physical volume metadata:

   ```
   pvremove <excluded-device>
   ```

i If logical volumes remain during the removal process, delete them in advance using the `lvremove` command. To protect against unintentional deletion, you can use the annotation `storage.deckhouse.io/deletion-protection` — until it is removed, the LVMVolumeGroup resource will not be deleted.

Protecting against data leaks between volumes

When files are deleted, the operating system does not erase the data on the disk: the corresponding blocks are merely marked as “free”. If the same physical blocks are used when creating a new volume, they may contain residual data from the previous user.

For example:

- User #1 created a volume from StorageClass #1 on node #1 (regardless of the “Block” or “Filesystem” mode) and wrote files to it.
- User #1 deleted the files and the volume itself.
- The blocks that the volume occupied were marked as free but not overwritten.
- User #2 requested a new volume from StorageClass #1 on the same node #1 in block mode.
- There is a risk that some or all of the previously freed blocks will be allocated to user #2 again.
- As a result, user #2 may gain access to user #1’s data.

Thick volume

To ensure confidentiality, the `volumeCleanup` parameter is used in the `PersistentVolume` configuration. Available modes:

- Parameter not set: When the volume is deleted, no additional operations are performed, the blocks remain uninitialized, and the data may be accessible to the next user.
- `RandomFillSinglePass`: The volume is overwritten once with random data (it is recommended to avoid this on solid-state drives due to wear).
- `RandomFillThreePass`: The volume is overwritten three times with random data (maximum reliability but high load on the device).
- `Discard`: Using TRIM, all block devices are marked as free. Effective on most modern solid-state drives, but depends on the SSD controller’s support for DRAT/RZAT.

Using Discard mode

Most modern solid-state drives support the `discard` (TRIM) command, which marks freed blocks as “zero” and does not return old data on subsequent reads. Therefore, using the `Discard` option is the most reliable way to prevent data leaks on solid-state drives. However, the following nuances should be

considered:

- The `discard` command initiates cell cleaning, but the actual zeroing is performed by the drive's controller in the background. Until this operation is completed, some blocks may still contain old data.
- Not all solid-state drives clean specific physical blocks individually: many devices work at the page level and larger blocks, so individual logical blocks may remain uncleaned until the entire page is fully processed.
- To be sure that after `discard` the blocks are indeed read as zeros, the disk must support DRAT (Deterministic TRIM) — predictable behavior of the `discard` command and RZAT (Deterministic Read Zero after TRIM) — guaranteed reading of "zeros" after `discard`. If the device has not confirmed support for DRAT and RZAT, there is no guarantee that the freed blocks will never return the old data.
- Even with declared support for DRAT/RZAT, manufacturers may not fully comply with the specifications. In the absence of independent tests or certification of solid-state drives, it is recommended to refrain from using the `Discard` mode for critical tasks related to data security.

⚠ If the drive is not confirmed to support DRAT and RZAT (meaning, has not passed verification for deterministic cleaning and reading of zeroed blocks), the use of the `Discard` option is **not recommended**, as there remains a risk of recovering old data.

Thin volume

When a block of a thin volume is released, the `discard` command from the guest OS is passed directly to the storage device. If a mechanical hard drive or a solid-state drive without `discard` support is used, the freed blocks in the thin pool remain with the previous data until they are physically overwritten. However, clients are provided only with thin volumes, not access to the thin pool itself. When a block is next allocated to a thin-LV, LVM guaranteedly fills the corresponding block in the thin pool with zeros (zero-fill), thanks to the `thin_pool_zero=1` parameter. This ensures the absence of residual data between different thin volumes and prevents information leaks between clients.

Additional resources

- Module `sds-node-configurator` custom resources

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration scenarios

i Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

Cloning virtual machines

When creating virtual machines using cloning methods, replace the UUID of the volume groups (VG) by executing:

```
vgchange -u
```

This command will generate new UUIDs for all VGs on the virtual machine. If necessary, the command can be added to the `cloud-init` script.

⚠ Changing the UUID is only possible if there are no active logical volumes (LV) in the group. They can be deactivated as follows:

```
lvchange -an <VG_or_LV_NAME>
```

Where `<VG_or_LV_NAME>` is the name of the VG to deactivate all volumes in the group, or the name of the LV to deactivate a specific volume.

Methods and scenarios for configuring the disk subsystem of nodes

The disk subsystem of each node can be organized in two ways, depending on whether the disks installed in the server are identical.

- Storage with identical disks: All disks in the node are of the same type and size.
- Combined storage: The node has disks of different types (for example, SSD + HDD).

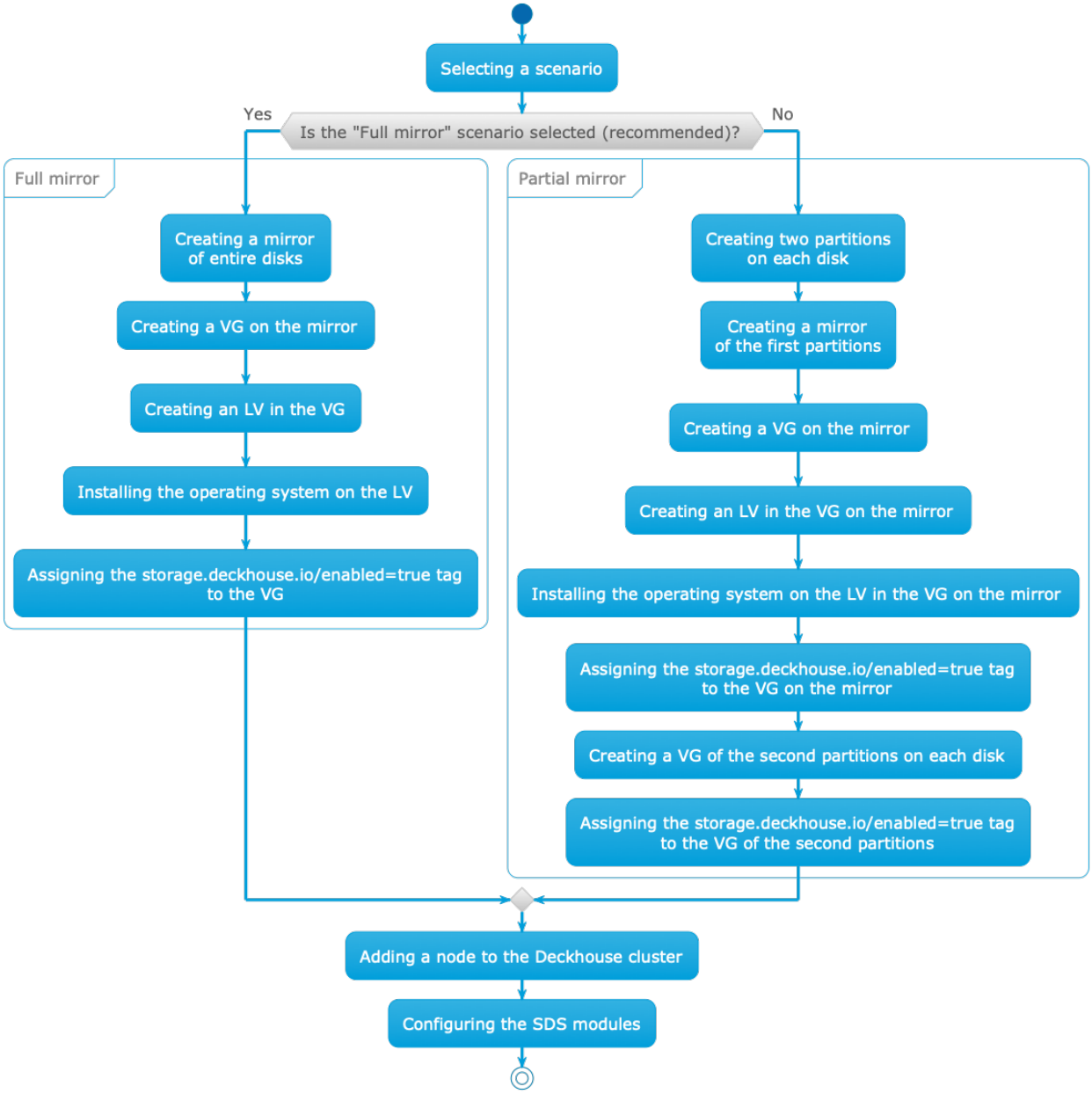
For each method of configuring the disk subsystem on nodes, there are two configuration scenarios:

- **Full mirror**: Recommended, reliable, and the simplest.
- **Partial mirror**: More flexible but requires caution.

The features, advantages, and disadvantages of the scenarios are presented in the table:

Configuration scenario	Implementation features	Advantages	Disadvantages
Full mirror	• Disks are not partitioned; a mirror is created from entire disks • Single VG for OS and data	• High reliability • Ease of setup and operation • Flexible resource allocation between SDS	• Excessive disk consumption with SDS replication
Partial mirror	• Disks are divided into 2 partitions • A mirror is created from the first partitions of each disk, on which a VG for the OS is created • A VG for data is created from the second partitions without mirroring	• Reliable storage • Maximum efficiency in space utilization	• Complex setup • Difficulties in reallocating space between safe and unsafe partitions

The differences in the configuration order of the disk subsystem depending on the chosen configuration scenario are illustrated in the diagram:



Storage with identical disks

Full mirror

We recommend using this configuration scenario as it is sufficiently reliable and easy to set up.

To configure a node according to this scenario, follow these steps:

1. Create a mirror from all disks (hardware or software). This mirror will be used simultaneously for the root system and for data.
2. Install the operating system:
   - Create a VG named `main` on the mirror.
   - Create an LV named `root` in the VG `main`.
   - Install the operating system on the LV `root`.
3. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main` using the following command:

   ```
   vgchange main --addtag storage.deckhouse.io/enabled=true
   ```

4. Add the prepared node to the Deckhouse cluster.

   If the node matches the `nodeSelector` specified in `spec.nodeSelector` of the `sds-replicated-volume` or `sds-local-volume` modules, the `sds-node-configurator` agent will detect the VG `main` and create a LVMVolumeGroup resource. This can be used in the `sds-local-volume` and `sds-replicated-volume` modules.

Example of configuring SDS modules (identical disks, “Full mirror”)

In this scenario, three nodes of the Deckhouse cluster are configured in “Full mirror” mode.
After automatic discovery, three CRD resources of type LVMVolumeGroup with automatically generated names will appear in the cluster.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be the following list:

```
NAME                                     THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG     AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                main   61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                main   4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                main   108s
```

Configuring the sds-local-volume module (identical disks, “Full mirror”)

To configure `sds-local-volume` in “Full Mirror” mode, create a LocalStorageClass resource and specify all discovered LVMVolumeGroup resources in it. This ensures that the VG with the label `main` is available on each node in the module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Configuring the sds-replicated-volume module (identical disks, “Full mirror”)

To configure the `sds-replicated-volume` module according to the “Full mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource and add all LVMVolumeGroup resources to it, so that the VG `main` is used on all nodes in the `sds-replicated-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

2. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resource in the `storagePool` field:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Partial mirror

⚠

- Do not use partitions with the same `PARTUUID`.
- Changing the `PARTUUID` of a partition on which a VG is already created is not supported.
- It is recommended to use GPT for the partition table — in MBR, `PARTUUID` is pseudo-random and contains the partition number, and there is no support for `PARTLABEL`, which may be useful for identifying partitions in Deckhouse.

In this scenario, two partitions are used on each disk:

- A partition for the root system and storage of SDS data that is not replicated.
- A partition for SDS data that is replicated.

The first partition of each disk is used to create a mirror, and the second is used to create a separate VG without mirroring. This allows for the most efficient use of disk space.

To configure a node according to the “Partial mirror” scenario, follow these steps:

1. During the operating system installation:
   - Create two partitions on each disk.
   - Assemble a mirror from the first partitions on each disk.
   - Create a VG named `main-safe` on the mirror.
   - Create an LV named `root` in the VG `main-safe`.
   - Install the operating system on the LV `root`.
2. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main-safe` using the following command:

   ```
   vgchange main-safe --addtag storage.deckhouse.io/enabled=true
   ```

3. Create a VG named `main-unsafe` from the second partitions of each disk.
4. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main-unsafe` using the following command:

   ```
   vgchange main-unsafe --addtag storage.deckhouse.io/enabled=true
   ```

5. Add the prepared node to the Deckhouse cluster.

   If the node matches the `nodeSelector` specified in `spec.nodeSelector` of the `sds-replicated-volume` or `sds-local-volume` modules, the `sds-node-configurator` agent will run on this node, detect the VGs `main-safe` and `main-unsafe`, and add the corresponding LVMVolumeGroup resources to the Deckhouse cluster. These LVMVolumeGroup resources can then be used to create volumes in the `sds-replicated-volume` or `sds-local-volume` modules.

Example of configuring SDS modules (identical disks, “Partial mirror”)

In this example, it is assumed that you have configured three nodes according to the “Partial mirror” scenario. In the Deckhouse cluster, six LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be the following list:

```
NAME                                   THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG            AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                main-safe     61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                main-safe     4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                main-safe     108s
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91   0/0         True                    Ready
worker-2   25596Mi   0                main-unsafe   61s
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c   0/0         True                    Ready
worker-0   25596Mi   0                main-unsafe   4m17s
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2   0/0         True                    Ready
worker-1   25596Mi   0                main-unsafe   108s
```

Configuring the sds-local-volume module (identical disks, “Partial mirror”)

To configure the `sds-local-volume` module according to the “Partial mirror” scenario, create a LocalStorageClass resource and add the LVMVolumeGroup resources to it so that only the VG `main-`

`safe` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Configuring the sds-replicated-volume module (identical disks, “Partial mirror”)

To configure the `sds-replicated-volume` module according to the “Partial mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource named `data-safe` and add the LVMVolumeGroup resources to it so that only the VG `main-safe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: None`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-safe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

2. Create a ReplicatedStoragePool resource named `data-unsafe` and add the LVMVolumeGroup resources to it so that only the VG `main-unsafe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: Availability` or `replication: ConsistencyAndAvailability`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-unsafe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
       - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
       - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
   EOF
   ```

3. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resources in the `storagePool` field so that the VGs `main-safe` and `main-unsafe` are used on all nodes:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data-safe # Note that due to replication: None, this resource
uses data-safe; therefore, data replication for persistent volumes (PV) created
with this StorageClass will not be performed.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data-unsafe # Note that due to replication: Availability, this
resource uses data-unsafe; therefore, data replication for PVs created with
this StorageClass will be performed.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data-unsafe # Note that due to replication:
ConsistencyAndAvailability, this resource uses data-unsafe; therefore, data
replication for PVs created with this StorageClass will be performed.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Combined storage

Combined storage involves the simultaneous use of disks of different types on a node.

When combining disks of different types to create storage, we recommend creating a mirror from disks of one type and installing the operating system on it according to the “Full mirror” scenario, but not using it for SDS.

For SDS, use disks of other types (hereinafter referred to as additional disks) that differ from those used for the mirror under the operating system.

Recommendations for using additional disks depending on their type:

Disk type	Recommended use cases
NVMe SSD	Creating volumes that require high performance
SATA SSD	Creating volumes that do not require high performance
HDD	Creating volumes that do not require high performance

Additional disks can be configured according to either the “Full mirror” or “Partial mirror” scenarios.

Below, the process of configuring additional disks will be considered using the following types as

examples:

- NVMe SSD
- SATA SSD
- HDD

Configuring additional disks (Full mirror)

⚠ The following describes the procedure for configuring additional disks for the case of initial deployment and configuration of the cluster when connecting to nodes via SSH. If you already have a running cluster and are adding additional disks to its nodes, it is recommended to create and configure VGs using the LVMVolumeGroup resource instead of executing the commands below on the node.

To configure additional disks on a node according to the “Full mirror” scenario, follow these steps:

1. Assemble a mirror from all additional disks of a certain type entirely (hardware or software).
2. Create a VG named `<vg-name>` on the mirror.
3. Set the tag `storage.deckhouse.io/enabled=true` for the VG `<vg-name>` using the following command:

   ```
   vgchange <vg-name> --addtag storage.deckhouse.io/enabled=true
   ```

i In the example above, replace `<vg-name>` with an informative name depending on the type of additional disks.

Examples of VG names for additional disks of different types:

- `ssd-nvme`: For NVMe SSD disks.
- `ssd-sata`: For SATA SSD disks.
- `hdd`: For HDD disks.

Example of configuring SDS modules (combined storage, “Full mirror”)

In this example, it is assumed that you have configured three nodes according to the “Full Mirror” scenario. In the Deckhouse cluster, three LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be a list like this:

```
NAME                                     THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG          AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>   61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>   4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>   108s
```

Where `<vg-name>` is the name assigned to the VG on the mirror in the previous step.

Configuring the sds-local-volume module (combined storage, “Full mirror”)

To configure the `sds-local-volume` module according to the “Full Mirror” scenario, create a LocalStorageClass resource and add all LVMVolumeGroup resources to it so that the VG `<vg-name>` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i In the example above, replace `<local-storage-class-name>` with an informative name depending on the type of additional disks.

Examples of informative names for the LocalStorageClass resource for additional disks of different types:

- `local-sc-ssd-nvme`: For NVMe SSD disks.
- `local-sc-ssd-sata`: For SATA SSD disks.
- `local-sc-hdd`: For HDD disks.

Configuring the sds-replicated-volume module (combined storage, “Full mirror”)

To configure the `sds-replicated-volume` module according to the “Full Mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource and add all LVMVolumeGroup resources to it so that the VG `<vg-name>` is used on all nodes in the `sds-replicated-volume` module:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: <replicated-storage-pool-name>
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

   i In the example above, replace `<replicated-storage-pool-name>` with an informative name depending on the type of additional disks.

   Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

   - `data-ssd-nvme`: For NVMe SSD disks.
   - `data-ssd-sata`: For SATA SSD disks.
   - `data-hdd`: For HDD disks.

2. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resource in the `storagePool` field:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: <replicated-storage-pool-name>
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: <replicated-storage-pool-name>
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: <replicated-storage-pool-name>
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Configuring additional disks (Partial mirror)

⚠
- Do not use partitions with the same `PARTUUID`.
- Changing the `PARTUUID` of a partition on which a VG is already created is not supported.
- It is recommended to use GPT for the partition table — in MBR, `PARTUUID` is pseudo-random and contains the partition number, and there is no support for `PARTLABEL`, which may be useful for identifying partitions in Deckhouse.

⚠ The following describes the procedure for configuring additional disks for the case of initial deployment and configuration of the cluster when connecting to nodes via SSH. If you already have a running cluster and are adding additional disks to its nodes, it is recommended to create and configure VGs using the LVMVolumeGroup resource instead of executing the commands below on the node.

In this scenario, two partitions are used on each disk: one for storing SDS data that is not replicated and another for SDS data that is replicated. The first partition of each disk is used to create a mirror, and the second is used to create a separate VG without mirroring. This allows for the most efficient use of disk space.

To configure a node with additional disks according to the “Partial mirror” scenario, follow these steps:

1. Create two partitions on each additional disk.
2. Assemble a mirror from the first partitions on each disk.
3. Create a VG named `<vg-name>-safe` on the mirror.
4. Create a VG named `<vg-name>-unsafe` from the second partitions of each disk.
5. Set the tag `storage.deckhouse.io/enabled=true` for the VGs `<vg-name>-safe` and `<vg-name>-unsafe` using the following commands:

```
vgchange <vg-name>-safe --addtag storage.deckhouse.io/enabled=true
vgchange <vg-name>-unsafe --addtag storage.deckhouse.io/enabled=true
```

> **i** In the example above, replace `<vg-name>` with an informative prefix depending on the type of additional disks.
>
> Examples of informative prefixes `<vg-name>` for additional disks of different types:
>
> - `ssd-nvme`: For NVMe SSD disks.
> - `ssd-sata`: For SATA SSD disks.
> - `hdd`: For HDD disks.

Example of configuring SDS modules (Combined storage, “Partial mirror”)

In this example, it is assumed that you have configured three nodes according to the “Partial Mirror” scenario. In the Deckhouse cluster, six LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be a list like this:

```
NAME                                   THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG                AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>-safe    61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>-safe    4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>-safe    108s
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>-unsafe  61s
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>-unsafe  4m17s
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>-unsafe  108s
```

Where `<vg-name>` is the prefix of the name assigned to the VGs created in the previous step.

Configuring the sds-local-volume module (combined storage, “Partial mirror”)

To configure the `sds-local-volume` module according to the “Partial Mirror” scenario, create a LocalStorageClass resource and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-safe` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i In the example above, replace `<local-storage-class-name>` with an informative name depending on the type of additional disks.

Examples of informative names for the LocalStorageClass resource for additional disks of different types:

- `local-sc-ssd-nvme`: For NVMe SSD disks.
- `local-sc-ssd-sata`: For SATA SSD disks.
- `local-sc-hdd`: For HDD disks.

Configuring the sds-replicated-volume module (combined storage, “Partial mirror”)

To configure the `sds-replicated-volume` module according to the “Partial mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource named `data-<vg-name>-safe` and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-safe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: None`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-<vg-name>-safe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

   i In the example above, replace `data-<vg-name>-safe` with an informative name depending on the type of additional disks.

   Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

   - `data-ssd-nvme-safe`: For NVMe SSD disks.
   - `data-ssd-sata-safe`: For SATA SSD disks.
   - `data-hdd-safe`: For HDD disks.

2. Create a ReplicatedStoragePool resource named `data-<vg-name>-unsafe` and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-unsafe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: Availability` or `replication: ConsistencyAndAvailability`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-<vg-name>-unsafe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
    - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
    - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
EOF
```

> **i** In the example above, replace `data-<vg-name>-unsafe` with an informative name depending on the type of additional disks.
>
> Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:
>
> - `data-ssd-nvme-unsafe`: For NVMe SSD disks.
> - `data-ssd-sata-unsafe`: For SATA SSD disks.
> - `data-hdd-unsafe`: For HDD disks.

3. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resources in the `storagePool` field so that the VGs `<vg-name>-safe` and `<vg-name>-unsafe` are used on all nodes:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: data-<vg-name>-safe # Note that due to replication: None, this
resource uses data-<vg-name>-safe; therefore, data replication for PVs created
with this StorageClass will not be performed.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: data-<vg-name>-unsafe # Note that due to replication:
Availability, this resource uses data-<vg-name>-unsafe; therefore, data
replication for PVs created with this StorageClass will be performed.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: data-<vg-name>-unsafe # Note that due to replication:
ConsistencyAndAvailability, this resource uses data-<vg-name>-unsafe;
therefore, data replication for PVs created with this StorageClass will be
performed.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

i

In the example above, replace `data-<vg-name>-unsafe` with an informative name depending on the type of additional disks.

Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

- `data-ssd-nvme-unsafe`: For NVMe SSD disks.
- `data-ssd-sata-unsafe`: For SATA SSD disks.
- `data-hdd-unsafe`: For HDD disks.

In a similar way, replace `data-<vg-name>-safe`.

Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

- `data-ssd-nvme-safe`: For NVMe SSD disks.
- `data-ssd-sata-safe`: For SATA SSD disks.
- `data-hdd-safe`: For HDD disks.

Additional resources

- Module `sds-local-volume` documentation
- Module `sds-node-configurator` documentation

- Module `sds-replicated-volume` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

Reasons why BlockDevice resources aren’t created in the cluster

Most often, BlockDevice resources are not created because the available devices do not pass the controller’s filters. Ensure that the devices meet the selection criteria.

Reasons why LVMVolumeGroup resources aren’t created in the cluster

- Absence of BlockDevice: The controller will not create an LVMVolumeGroup if there are no BlockDevice resources in the cluster specified in its specification.
- Absence of a tag: If BlockDevice resources are present but the LVMVolumeGroup is missing, check that the corresponding LVM group on the node has the tag `storage.deckhouse.io/enabled=true`.

Reasons why after deleting the LVMVolumeGroup, the resource and Volume Group remain

This situation is possible in two cases:

1. There are logical volumes in the Volume Group — the controller is not responsible for deleting logical volumes on the node, so if there are any logical volumes in the Volume Group created via the resource, they must be manually deleted. After that, both the LVMVolumeGroup resource and the Volume Group itself (including physical volumes) will be automatically deleted.
2. The resource has the annotation `storage.deckhouse.io/deletion-protection` — this annotation protects the resource from deletion, along with the associated Volume Group. Remove the annotation with the command:

   ```
   d8 k annotate lvg <resource-name> storage.deckhouse.io/deletion-protection-
   ```

   After that, the LVMVolumeGroup resource and the corresponding Volume Group will be automatically deleted.

Reasons for failed attempts to create a Volume Group using the LVMVolumeGroup resource

Most likely, the resource did not pass validation by the controller (unlike the Kubernetes schema). The reason can be found in the `status.message` field of the LVMVolumeGroup resource or in the controller logs. Check that the specified BlockDevice resources meet the conditions:

- The `consumable` field is set to `true`.
- For `spec.type: Local`, all BlockDevice resources belong to the same node.
- Current names of BlockDevice resources are used.

Behavior of the LVMVolumeGroup resource when one of the devices in the Volume Group is disabled

The LVMVolumeGroup resource remains as long as the corresponding LVM group exists. When a device becomes unavailable, the group enters an error state, which is reflected in the `status` field.

After the device is restored, the group returns to the `Healthy` status, and the resource status is

automatically updated.

Transferring management of an existing Volume Group on the node to the controller

Add the tag `storage.deckhouse.io/enabled=true` to the desired Volume Group:

```
vgchange myvg-0 --addtag storage.deckhouse.io/enabled=true
```

The controller will create the corresponding LVMVolumeGroup resource and take over management of the group.

Disabling tracking of the LVM Volume Group by the controller

Remove the tag `storage.deckhouse.io/enabled=true`:

```
vgchange myvg-0 --deltag storage.deckhouse.io/enabled=true
```

The controller will stop tracking and delete the associated LVMVolumeGroup resource.

Reasons for automatic setting of the tag storage.deckhouse.io/enabled=true on the Volume Group

The controller adds the tag when creating the Volume Group through the LVMVolumeGroup resource.

When migrating from the `linstor` module to `sds-node-configurator` and `sds-replicated-volume`, all `linstor-*` tags are replaced with `storage.deckhouse.io/enabled=true` to transfer management to the new logic.

Using the LVMVolumeGroupSet resource to create LVMVolumeGroup

The LVMVolumeGroupSet resource allows template-based creation of LVMVolumeGroup on nodes. Currently, the PerNode strategy is supported — one resource per node that matches the selector.

Example specification of LVMVolumeGroupSet:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroupSet
metadata:
  name: my-lvm-volume-group-set
  labels:
    my-label: my-value
spec:
  strategy: PerNode
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
  lvmVolumeGroupTemplate:
    metadata:
      labels:
        my-label-for-lvg: my-value-for-lvg
    spec:
      type: Local
      blockDeviceSelector:
        matchLabels:
          status.blockdevice.storage.deckhouse.io/model: <model>
      actualVGNameOnTheNode: <VG-name-on-the-node>
```

Labels added by the controller to BlockDevice resources

- `status.blockdevice.storage.deckhouse.io/type`: LVM type.
- `status.blockdevice.storage.deckhouse.io/fstype`: File system type.
- `status.blockdevice.storage.deckhouse.io/pvuuid`: Physical volume (PV) UUID.

- `status.blockdevice.storage.deckhouse.io/vguuid`: Volume group (VG) UUID.
- `status.blockdevice.storage.deckhouse.io/partuuid`: Partition UUID.
- `status.blockdevice.storage.deckhouse.io/lvmvolumegroupname`: Name of the LVMVolumeGroup resource.
- `status.blockdevice.storage.deckhouse.io/actualvgnameonthenode`: Name of the LVM Volume Group on the node.
- `status.blockdevice.storage.deckhouse.io/wwn`: WWN (World Wide Name) of the device.
- `status.blockdevice.storage.deckhouse.io/serial`: Serial number of the device.
- `status.blockdevice.storage.deckhouse.io/size`: Size of the device.
- `status.blockdevice.storage.deckhouse.io/model`: Model of the device.
- `status.blockdevice.storage.deckhouse.io/rota`: Rotational device flag.
- `status.blockdevice.storage.deckhouse.io/hotplug`: Hotplug capability.
- `status.blockdevice.storage.deckhouse.io/machineid`: Identifier of the machine where the device is installed.

Additional resources

- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Setting up local storage based on LVM

Local storage reduces network latency and provides higher performance compared to remote storage accessed over a network. This approach is particularly effective in test environments and EDGE clusters. This functionality is provided by the `sds-local-volume` module.

Configuring local storage

To ensure the correct operation of the `sds-local-volume` module, follow these steps:

1. Configure LVMVolumeGroup. Before creating a StorageClass, you must create an LVMVolumeGroup resource for the `sds-node-configurator` module on the cluster nodes.
2. Enable the `sds-node-configurator` module. Ensure that the module is enabled **before** enabling the `sds-local-volume` module.
3. Create the corresponding StorageClasses. Creating a StorageClass for the CSI driver `local.csi.storage.deckhouse.io` by a user is **prohibited**.

The module supports two operating modes: LVM and LVMThin.

Quick start

All commands are executed on a machine with access to the Kubernetes API and administrator privileges.

Enabling modules

Enabling the `sds-node-configurator` module:

1. Create a ModuleConfig resource to enable the module:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

2. Wait for the module to reach the `Ready` state. At this stage, it is not necessary to check the pods in the `d8-sds-node-configurator` namespace.

```
d8 k get modules sds-node-configurator -w
```

Enabling the `sds-local-volume` module:

1. Activate the `sds-local-volume` module. The example below starts the module with default settings, which will create service pods for the `sds-local-volume` component on all cluster nodes:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  version: 1
EOF
```

2. Wait for the module to reach the `Ready` state.

```
d8 k get modules sds-local-volume -w
```

3. Ensure that all pods in the `d8-sds-local-volume` and `d8-sds-node-configurator` namespaces are in the `Running` or `Completed` state and are running on all nodes where LVM resources are planned to be used.

```
d8 k -n d8-sds-local-volume get pod -owide -w
d8 k -n d8-sds-node-configurator get pod -o wide -w
```

Preparing nodes for storage creation

For storage to function correctly on nodes, the `sds-local-volume-csi-node` pods must be running on the selected nodes.

By default, these pods are launched on all cluster nodes. You can verify their presence using the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

The placement of `sds-local-volume-csi-node` pods is managed by specific labels (`nodeSelector`). These labels are set in the `spec.settings.dataNodes.nodeSelector` parameter of the module.

Configuring storage on nodes

To configure storage on nodes, you need to create LVM volume groups using LVMVolumeGroup resources. This example creates a Thick storage.

⚠ Before creating an LVMVolumeGroup resource, ensure that the `sds-local-volume-csi-node` pod is running on the respective node. This can be checked with the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

1. Retrieve all BlockDevice resources available in your cluster:

```
d8 k get bd
```

Example output:

```
NAME                                            NODE       CONSUMABLE   SIZE
PATH
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa   worker-0   false
976762584Ki    /dev/nvme1n1
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd   worker-0   false
894006140416   /dev/nvme0n1p6
dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0   worker-1   false
976762584Ki    /dev/nvme1n1
dev-b103062f879a2349a9c5f054e0366594568de68d   worker-1   false
894006140416   /dev/nvme0n1p6
dev-53d904f18b912187ac82de29af06a34d9ae23199   worker-2   false
976762584Ki    /dev/nvme1n1
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1   worker-2   false
894006140416   /dev/nvme0n1p6
```

2. Create an LVMVolumeGroup resource for the `worker-0` node:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-0" # The name can be any fully qualified resource name
in Kubernetes. This LVMVolumeGroup resource name will be used to create
LocalStorageClass in the future.
spec:
  type: Local
  local:
    nodeName: "worker-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
          - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
  actualVGNameOnTheNode: "vg-1" # The name of the LVM VG that will be created
from the specified block devices on the node.
EOF
```

3. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

```
d8 k get lvg vg-1-on-worker-0 -W
```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-0` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

4. Create an LVMVolumeGroup resource for the `worker-1` node:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-1"
spec:
  type: Local
  local:
    nodeName: "worker-1"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0
          - dev-b103062f879a2349a9c5f054e0366594568de68d
  actualVGNameOnTheNode: "vg-1"
EOF
```

5. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

   ```
   d8 k get lvg vg-1-on-worker-1 -w
   ```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-1` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

6. Create an LVMVolumeGroup resource for the `worker-2` node:

   ```
   d8 k apply -f - <<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: LVMVolumeGroup
   metadata:
     name: "vg-1-on-worker-2"
   spec:
     type: Local
     local:
       nodeName: "worker-2"
     blockDeviceSelector:
       matchExpressions:
         - key: kubernetes.io/metadata.name
           operator: In
           values:
             - dev-53d904f18b912187ac82de29af06a34d9ae23199
             - dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1
     actualVGNameOnTheNode: "vg-1"
   EOF
   ```

7. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

   ```
   d8 k get lvg vg-1-on-worker-2 -w
   ```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-2` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

8. Create a LocalStorageClass resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

For a thin volume:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
        thin:
          poolName: thin-1
    type: Thin
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Important. In a LocalStorageClass with `type: Thick`, you cannot use an LVMVolumeGroup that contains at least one thin pool.

9. Wait for the created LocalStorageClass resource to transition to the `Created` state:

```
d8 k get lsc local-storage-class -w
```

10. Verify that the corresponding StorageClass has been created:

```
d8 k get sc local-storage-class
```

If a StorageClass named `local-storage-class` appears, the configuration of the `sds-local-volume` module is complete. Users can now create PVCs by specifying the StorageClass named `local-storage-class`.

Selecting a volume cleanup method after PV deletion

When files are deleted, the operating system does not physically erase the content but only marks the corresponding blocks as “free.” If a new volume receives physical blocks previously used by another volume, those blocks may still contain the previous user’s data.

This is possible, for example, in the following scenario:

- User #1 places files in a volume requested from StorageClass 1 on node 1 (in either “Block” or “Filesystem” mode).
- User #1 deletes the files and the volume.

- The physical blocks that were occupied become “free” but are not wiped.
- User #2 requests a new volume from StorageClass 1 on node 1 in “Block” mode.
- There is a risk that some or all of the blocks previously used by User #1 will be reallocated to User #2.
- In this case, User #2 may be able to recover User #1’s data.

Thick volumes

To prevent data leaks through thick volumes, the `volumeCleanup` parameter is provided. It allows you to select the volume cleanup method before deleting a PV.

Possible values:

- Parameter not set: No additional actions are performed when deleting the volume. Data may remain accessible to the next user.
- `RandomFillSinglePass`: The volume is overwritten with random data once before deletion. This option is not recommended for solid-state drives, as it reduces the drive’s lifespan.
- `RandomFillThreePass`: The volume is overwritten with random data three times before deletion. This option is not recommended for solid-state drives, as it reduces the drive’s lifespan.
- `Discard`: All volume blocks are marked as free using the `discard` system call before deletion. This option is only meaningful for solid-state drives.

  Most modern solid-state drives guarantee that a block marked with `discard` will not return previous data when read. This makes the `Discard` option the most effective way to prevent data leaks when using solid-state drives.

  However, erasing a cell is a relatively slow operation, so it is performed by the device in the background. Additionally, many drives cannot erase individual cells but only groups (pages). As a result, not all drives guarantee the immediate unavailability of freed data. Furthermore, not all drives that claim to guarantee this keep their promise. If a device does not guarantee Deterministic TRIM (DRAT), Deterministic Read Zero after TRIM (RZAT), and is not verified, it is not recommended for use.

Thin volumes

When a block in a thin volume is freed via `discard` by the guest operating system, this command is forwarded to the device. If a hard disk is used or if the solid-state drive does not support `discard`, the data may remain in the thin pool until the block is reused. However, users are only granted access to thin volumes, not the thin pool itself. They can only obtain a volume from the pool, and for thin volumes, the thin pool block is zeroed upon reuse, preventing data leaks between clients. This is ensured by the `thin_pool_zero=1` setting in LVM.

System requirements and recommendations

- Use stock kernels provided with supported distributions.
- Do not use another SDS (Software Defined Storage) to provide disks for SDS Deckhouse.

Additional resources

- Module `sds-local-volume` documentation
- Module `sds-node-configurator` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing local storage based on LVM

Selecting specific nodes for module usage

To restrict the module’s usage to specific cluster nodes, you need to set labels in the `nodeSelector` field in the module settings.

To display and edit the module settings, execute the command:

```
d8 k edit mc sds-local-volume
```

Configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  settings:
    dataNodes:
      nodeSelector:
        my-custom-label-key: my-custom-label-value
status:
  message: ""
  version: "1"
```

To view the current labels in the `nodeSelector` field, use the following command:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

The module selects only those nodes that have all the labels specified in the `nodeSelector`. By modifying this field, you can control the list of nodes on which the module will run.

> **i** You can specify multiple labels in the `nodeSelector`. However, for the module to work correctly, all these labels must be present on each node where you intend to run `sds-local-volume-csi-node`.

After configuring the labels, ensure that the `sds-local-volume-csi-node` Pods are running on the target nodes. You can check their presence with the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

Verifying PVC creation on the selected node

Make sure that the `sds-local-volume-csi-node` Pod is running on the selected node. To do this, run the command:

```
d8 k -n d8-sds-local-volume get po -owide
```

If the Pod is absent, verify that all the labels specified in the module settings in the nodeSelector field are present on the node. For available solutions when Pods are missing from the target node, refer to this section.

Removing a node from the module management

To remove a node from module management, you need to delete the labels set in the `nodeSelector` field in the `sds-local-volume` module settings.

To check the current labels, execute the command:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Remove the specified labels from the nodes with the command:

```
d8 k label node %node-name% %label-from-selector%-
```

⚠ After the label key, you must specify a minus sign to remove it.

After this, the `sds-local-volume-csi-node` Pod should be removed from the node. Check its status with the command:

```
d8 k -n d8-sds-local-volume get po -owide
```

If the Pod remains after removing the label, ensure that the labels from the `d8-sds-local-volume-controller-config` config are actually removed. You can verify this using:

```
d8 k get node %node-name% --show-labels
```

If the labels are absent, check that the node does not have any LVMVolumeGroup resources being used by LocalStorageClass resources. More information on this check can be found in this section.

⚠ Note that for LVMVolumeGroup and LocalStorageClass resources that are the reason why the node cannot be removed from module management, the label `storage.deckhouse.io/sds-local-volume-candidate-for-eviction` will be assigned.

On the node itself, the label `storage.deckhouse.io/sds-local-volume-need-manual-eviction` will be present.

Verifying dependent LVMVolumeGroup resources on the node

To check the dependent resources, follow these steps:

1. Display the available LocalStorageClass resources:

   ```
   d8 k get lsc
   ```

2. Check each of them for the list of used LVMVolumeGroup resources.

   If you want to list all LocalStorageClass resources at once, run the command:

   ```
   d8 k get lsc -oyaml
   ```

   Example LocalStorageClass resource:

```
apiVersion: v1
items:
- apiVersion: storage.deckhouse.io/v1alpha1
  kind: LocalStorageClass
  metadata:
    finalizers:
    - localstorageclass.storage.deckhouse.io
    name: test-sc
  spec:
    lvm:
      lvmVolumeGroups:
      - name: test-vg
      type: Thick
    reclaimPolicy: Delete
    volumeBindingMode: WaitForFirstConsumer
  status:
    phase: Created
kind: List
```

Pay attention to the `spec.lvm.lvmVolumeGroups` field. It specifies the used LVMVolumeGroup resources.

3. Display the list of existing LVMVolumeGroup resources.

   ```
   d8 k get lvg
   ```

   Example LVMVolumeGroup output:

   ```
   NAME              HEALTH        NODE            SIZE       ALLOCATED SIZE   VG        AGE
   lvg-on-worker-0   Operational   node-worker-0   40956Mi    0                test-vg   15d
   lvg-on-worker-1   Operational   node-worker-1   61436Mi    0                test-vg   15d
   lvg-on-worker-2   Operational   node-worker-2   122876Mi   0                test-vg   15d
   lvg-on-worker-3   Operational   node-worker-3   307196Mi   0                test-vg   15d
   lvg-on-worker-4   Operational   node-worker-4   307196Mi   0                test-vg   15d
   lvg-on-worker-5   Operational   node-worker-5   204796Mi   0                test-vg   15d
   ```

4. Ensure that the node you intend to remove from the module's control does not have any LVMVolumeGroup resources used in LocalStorageClass resources. To avoid unintentionally losing control over volumes already created using the module, the user needs to manually delete dependent resources by performing necessary operations on the volume.

Remaining sds-local-volume-csi-node Pod after removing labels

If after removing the labels from the node the `sds-local-volume-csi-node` Pod continues to run, this is most likely due to the presence on the node of LVMVolumeGroup resources that are used by one of the LocalStorageClass resources. The verification process is described above.

Absence of component service Pods on the desired node

The issue may be related to incorrectly set labels. The nodes used by the module are determined by the labels specified in the module settings in the `nodeSelector` field. To view the current labels, run:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Additionally, you can check the selectors used by the module in the secret configuration `d8-sds-local-volume-controller-config` in the `d8-sds-local-volume` namespace:

```
d8 k -n d8-sds-local-volume get secret d8-sds-local-volume-controller-config  -o
jsonpath='{.data.config}' | base64 --decode
```

Example output:

```
nodeSelector:
  kubernetes.io/os: linux
  my-custom-label-key: my-custom-label-value
```

This command output should list all the labels from the module settings `data.nodeSelector`, as well as `kubernetes.io/os: linux`.

Check the labels on the desired node:

```
d8 k get node %node-name% --show-labels
```

If necessary, add the missing labels to the desired node:

```
d8 k label node %node-name% my-custom-label-key=my-custom-label-value
```

If the labels are present, check for the presence of the label `storage.deckhouse.io/sds-local-volume-node=` on the node. If this label is missing, ensure that the `sds-local-volume-controller` is running, and review its logs:

```
d8 k -n d8-sds-local-volume get po -l app=sds-local-volume-controller
d8 k -n d8-sds-local-volume logs -l app=sds-local-volume-controller
```

Data migration between PVCs

Copy the following script into a file named `migrate.sh` on any master node:

```
#!/bin/bash

ns=$1
src=$2
dst=$3

if [[ -z $3 ]]; then
  echo "You must give as args: namespace source_pvc_name destination_pvc_name"
  exit 1
fi

echo "Creating job yaml"
cat > migrate-job.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: migrate-pv-$src
  namespace: $ns
spec:
  template:
    spec:
      containers:
      - name: migrate
        image: debian
        command: [ "/bin/bash", "-c" ]
        args:
          -
            apt-get update && apt-get install -y rsync &&
            ls -lah /src_vol /dst_vol &&
            df -h &&
            rsync -avPS --delete /src_vol/ /dst_vol/ &&
            ls -lah /dst_vol/ &&
            du -shxc /src_vol/ /dst_vol/
        volumeMounts:
```

```
        - mountPath: /src_vol
          name: src
          readOnly: true
        - mountPath: /dst_vol
          name: dst
      restartPolicy: Never
      volumes:
      - name: src
        persistentVolumeClaim:
          claimName: $src
      - name: dst
        persistentVolumeClaim:
          claimName: $dst
  backoffLimit: 1
EOF

d8 k create -f migrate-job.yaml
d8 k -n $ns get jobs -o wide
kubectl_completed_check=0

echo "Waiting for data migration to be completed"
while [[ $kubectl_completed_check -eq 0 ]]; do
    d8 k -n $ns get pods | grep migrate-pv-$src
   sleep 5
   kubectl_completed_check=`d8 k -n $ns get pods | grep migrate-pv-$src | grep "Completed" |
wc -l`
done
echo "Data migration completed"
```

To use the script, run the following command:

```
migrate.sh NAMESPACE SOURCE_PVC_NAME DESTINATION_PVC_NAME
```

Creating volume snapshots

You can read more about snapshots and associated resources in the Kubernetes documentation.

1. Enable the `snapshot-controller` module:

   ```
   d8 k apply -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: snapshot-controller
   spec:
     enabled: true
     version: 1
   EOF
   ```

2. Now you can create volume snapshots. To do this, execute the following command with the necessary parameters:

```
d8 k apply -f -<<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: my-snapshot
  namespace: <name of the namespace where the PVC is located>
spec:
  volumeSnapshotClassName: sds-local-volume-snapshot-class
  source:
    persistentVolumeClaimName: <name of the PVC to snapshot>
EOF
```

Note that `sds-local-volume-snapshot-class` is created automatically, and its `deletionPolicy` is set to `Delete`, which means that the VolumeSnapshotContent resource will be deleted when the associated VolumeSnapshot resource is deleted.

3. To check the status of the created snapshot, execute the command:

   ```
   d8 k get volumesnapshot
   ```

   This command will display a list of all snapshots and their current status.

Additional resources

- Module `sds-local-volume` parameters
- Module `sds-local-volume` custom resources
- Module `sds-node-configurator` custom resources
- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Setting up replicated storage based on DRBD

Data replication across multiple nodes ensures fault tolerance and data availability, even if a hardware or software failure occurs on one of the nodes. This guarantees data preservation on other nodes, maintaining continuous access. Such a model is essential for critical data and distributed infrastructures requiring high availability and minimizing data loss during failures.

To create replicated block StorageClass objects based on Distributed Replicated Block Device (DRBD), the `sds-replicated-volume` module is used. It leverages LINSTOR as the backend.

Enabling the module

Discovery of LVM components

Before creating StorageClass objects based on Logical Volume Manager (LVM), it is necessary to detect the block devices and volume groups available on the nodes and obtain current information about their state. To do this, enable the `sds-node-configurator` module:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

Wait for the `sds-node-configurator` module to reach the `Ready` status. To check the status, run the following command:

```
d8 k get modules sds-node-configurator -w
```

In the output, you should see information about the `sds-node-configurator` module:

```
NAME                      STAGE   SOURCE     PHASE       ENABLED    READY
sds-node-configurator             Embedded   Available   True       True
```

Connecting DRBD

To enable the `sds-replicated-volume` module with default settings, run the following command:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

This will install the DRBD kernel module on all cluster nodes, register the CSI driver, and launch the `sds-replicated-volume` component Pods.

Wait until the `sds-replicated-volume` module reaches the `Ready` status. To check the module status, run the following command:

```
d8 k get modules sds-replicated-volume -w
```

In the output, you should see information about the `sds-replicated-volume` module:

```
NAME                      STAGE   SOURCE     PHASE       ENABLED    READY
sds-replicated-volume             Embedded   Available   True       True
```

To check that all Pods in the `d8-sds-replicated-volume` and `d8-sds-node-configurator` namespaces are in the `Running` or `Completed` state and have been started on all nodes where DRBD resources are planned to be used, use the following commands:

```
d8 k -n d8-sds-replicated-volume get pod -w
d8 k -n d8-sds-node-configurator get pod -w
```

> **i** Avoid configuring the `LINSTOR` backend manually, as this can can result in errors.

Node pre-configuration

Creating LVM volume groups

Before configuring the creation of StorageClass objects, you need to combine the available block devices on the nodes into LVM volume groups. These volume groups will later be used to place PersistentVolumes. To get the available block devices, you can use the BlockDevice resource, which reflects their current state:

```
d8 k get bd
```

In the output, you should see a list of available block devices:

```
NAME                                             NODE       CONSUMABLE   SIZE
PATH
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa   worker-0   false        976762584Ki
/dev/nvme1n1
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd   worker-0   false        894006140416
/dev/nvme0n1p6
dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0   worker-1   false        976762584Ki
/dev/nvme1n1
dev-b103062f879a2349a9c5f054e0366594568de68d   worker-1   false        894006140416
/dev/nvme0n1p6
dev-53d904f18b912187ac82de29af06a34d9ae23199   worker-2   false        976762584Ki
/dev/nvme1n1
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1   worker-2   false        894006140416
/dev/nvme0n1p6
```

In the example output, there are six block devices available across three nodes.

To combine the block devices on a node, you need to create an LVM volume group using the LVMVolumeGroup resource. To create the LVMVolumeGroup resource on node `worker-0`, apply the following resource, replacing the node and block device names with your own:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-on-worker-0"
spec:
  type: Local
  local:
    # Replace with the name of the node where you are creating the volume group.
    nodeName: "worker-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          # Replace with the names of the block devices for the node you are
creating the volume group on.
          - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
          - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
  # The name of the LVM volume group that will be created from the specified block
devices on the selected node.
  actualVGNameOnTheNode: "vg"
  # Uncomment if you want to be able to create thin pools.
  # thinPools:
  #   - name: thin-pool-0
  #     size: 70%
EOF
```

Wait for the created LVMVolumeGroup resource to reach the `Ready` phase. To check the resource phase, run the following command:

```
d8 k get lvg vg-on-worker-0 -w
```

In the output, you should see information about the resource phase:

```
NAME             THINPOOLS   CONFIGURATION APPLIED   PHASE   NODE       SIZE
ALLOCATED SIZE   VG   AGE
vg-on-worker-0   1/1         True                    Ready   worker-0   360484Mi
30064Mi          vg   1h
```

If the resource has transitioned to the `Ready` phase, this means that an LVM volume group named vg has been created on node `worker-0` from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

Next, you need to repeat the creation of LVMVolumeGroup resources for the remaining nodes (`worker-1` and `worker-2`), changing the resource name, node name, and block device names accordingly.

Ensure that LVM volume groups have been created on all nodes where they are planned to be used by running the following command:

```
d8 k get lvg -W
```

In the output, you should see a list of created volume groups:

```
NAME             THINPOOLS   CONFIGURATION APPLIED   PHASE   NODE       SIZE
ALLOCATED SIZE   VG   AGE
vg-on-worker-0   0/0         True                    Ready   worker-0   360484Mi
30064Mi          vg   1h
vg-on-worker-1   0/0         True                    Ready   worker-1   360484Mi
30064Mi          vg   1h
vg-on-worker-2   0/0         True                    Ready   worker-2   360484Mi
30064Mi          vg   1h
```

Creating replicated thick pools

> **i** To work with thin pool, you need to enable the parameter enableThinProvisioning

Now that the necessary LVM volume groups are created on the nodes, you need to combine them into a single logical space. This can be done by combining them into replicated storage pools in the `LINSTOR` backend through the ReplicatedStoragePool resource interface.

Storage pools can be of two types: LVM (thick) and LVMThin (thin). The thick pool offers high performance, comparable to the performance of the storage device, but it does not allow snapshots. Example of creating a replicated thick pool:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thick-pool
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
    - name: vg-1-on-worker-1
    - name: vg-1-on-worker-2
EOF
```

Wait for the created ReplicatedStoragePool resource to reach the `Completed` phase. To check the resource phase, run the following command:

```
d8 k get rsp data -W
```

In the output, you should see information about the resource phase:

```
NAME         PHASE       TYPE   AGE
thick-pool   Completed   LVM    87d
```

Creating replicated thin pools

Unlike thick pools, a thin pool allows the use of snapshots but has lower performance.

The previously created LVMVolumeGroup is suitable for creating thick pools. If you need the ability to

create replicated thin pools, update the configuration of the LVMVolumeGroup resources by adding a definition for the thin pool:

```
d8 k patch lvg vg-on-worker-0 --type='json' -p='[
  {
    "op": "add",
    "path": "/spec/thinPools",
    "value": [
      {
        "name": "thin-pool-0",
        "size": "70%"
      }
    ]
  }
]'
```

In the updated version of LVMVolumeGroup, 70% of the available space will be used to create thin pools. The remaining 30% can be used for thick pools.

Repeat the addition of thin pools for the remaining nodes (`worker-1` and `worker-2`). Example of creating a replicated thin pool:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thin-pool
spec:
  type: LVMThin
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-1
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-2
      thinPoolName: thin-pool-0
EOF
```

Wait for the created ReplicatedStoragePool resource to transition to the `Completed` phase. To check the resource phase, run the following command:

```
d8 k get rsp data -w
```

In the output, you should see information about the resource phase:

```
NAME        PHASE       TYPE      AGE
thin-pool   Completed   LVMThin   87d
```

Creating StorageClass objects

StorageClass objects are created through the ReplicatedStorageClass resource, which defines the configuration for the desired StorageClass. Manually creating a StorageClass resource without ReplicatedStorageClass may lead to undesired behavior.

Example of creating a ReplicatedStorageClass resource based on a thick pool, where the PersistentVolumes will be placed on volume groups across three nodes:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-storage-class
spec:
  # Specify the name of one of the storage pools created earlier.
  storagePool: thick-pool
  # Reclaim policy when deleting PVC.
  # Allowed values: "Delete", "Retain".
  # [Learn more...](https://kubernetes.io/docs/concepts/storage/persistent-
volumes/#reclaiming)
  reclaimPolicy: Delete
  # Replicas can be placed on any available node: no more than one replica of a
specific volume on one node.
  # The cluster does not have zones (no nodes with the topology.kubernetes.io/zone
labels).
  topology: Ignored
  # Replication mode where the volume remains available for read and write even if
one replica becomes unavailable.
  # Data is stored in three instances across different nodes.
  replication: ConsistencyAndAvailability
EOF
```

Check that the created ReplicatedStorageClass resource has transitioned to the `Created` phase by running the following command:

```
d8 k get rsc replicated-storage-class -w
```

In the output, you should see information about the created ReplicatedStorageClass resource:

```
NAME                        PHASE     AGE
replicated-storage-class    Created   1h
```

Check that the corresponding StorageClass has been generated by running the following command:

```
d8 k get sc replicated-storage-class
```

In the output, you should see information about the generated StorageClass:

```
NAME                        PROVISIONER                       RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
replicated-storage-class    local.csi.storage.deckhouse.io    Delete
WaitForFirstConsumer   true                   1h
```

If a StorageClass with the name `replicated-storage-class` appears, it means the configuration of the `sds-replicated-volume` module is complete. Users can now create PersistentVolume objects by specifying the `replicated-storage-class` StorageClass. With the above configuration, a volume with three replicas across different nodes will be created.

Additional resources

- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing DRBD-based replicated storage

⚠ Functionality is guaranteed only if the system requirements are met. Using it under other conditions is possible, but stable operation is not guaranteed.

System requirements

i Applicable to both single-zone clusters and clusters that use multiple availability zones.

- Use stock kernels shipped with the supported distributions.
- For network connectivity, use infrastructure with bandwidth of 10 Gbps or higher.
- To achieve maximum performance, network latency between nodes should be within 0.5–1 ms.
- Do not use another SDS (Software-defined storage) to provide disks for Deckhouse SDS.

Recommendations

- Do not use RAID. See below for details.
- Use local physical disks. See below.
- For stable cluster operation (with reduced performance), allowable network latency between nodes must not exceed 20 ms.

Retrieving space-usage information

There are two ways to obtain this information:

1. Via the Grafana dashboard:

   Go to “Dashboards” → “Storage” → “LINSTOR/DRBD”. The current cluster space usage level is shown in the upper-right corner.

   i **Warning.** The value reflects the state of all available space. When you create volumes with two replicas, divide the reported figures by two to estimate how many volumes can actually be placed.

2. Via the command line:

   ```
   d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list
   ```

   i **Warning.** The value reflects the state of all available space. When you create volumes with two replicas, divide the reported figures by two to estimate how many volumes can actually be placed.

Assigning the default StorageClass

To designate a StorageClass as the default, set the field `spec.isDefault` to `true` in the ReplicatedStorageClass custom resource.

Adding an existing LVMVolumeGroup

1. Assign the tag `storage.deckhouse.io/enabled=true` to the Volume Group:

   ```
   vgchange myvg-0 --add-tag storage.deckhouse.io/enabled=true
   ```

   After that the Volume Group will be discovered automatically and a corresponding LVMVolumeGroup resource will be created.

2. Reference this resource in ReplicatedStoragePool parameters using `spec.lvmVolumeGroups[].name`.
   If an LVMThin pool is used, additionally specify its name in `spec.lvmVolumeGroups[].thinPoolName`.

Changing DRBD volume limits and cluster ports

The default port range for DRBD resources is TCP `7000`–`7999`. You can override it with the `drbdPortRange` setting by specifying the desired `minPort` and `maxPort` values.

⚠ After changing `drbdPortRange`, restart the LINSTOR controller so that the new settings take effect. Existing DRBD resources will retain their assigned ports.

Correctly rebooting a node with DRBD resources

i To ensure stable module operation, avoid rebooting multiple nodes at the same time.

1. Drain the target node:

   ```
   d8 k drain test-node-1 --ignore-daemonsets --delete-emptydir-data
   ```

2. Make sure there are no faulty DRBD resources or resources in the `SyncTarget` state. To check this, run the following command and analyze the output:

   ```
   d8 k -n d8-sds-replicated-volume exec -t deploy/linstor-controller -- linstor r l --faulty
   ```

   If any resources are in the `SyncTarget` state, wait for the synchronization to complete or take corrective actions.

   Example output:

   ```
   Defaulted container "linstor-controller" out of: linstor-controller, kube-rbac-
   proxy
   +-------------------------------------------------------------------+
   | ResourceName | Node | Port | Usage | Conns | State | CreatedOn |
   |===================================================================|
   +-------------------------------------------------------------------+
   ```

3. Reboot the node and wait for all DRBD resources to resynchronize, then run `uncordon`:

   ```
   d8 k uncordon test-node-1
   node/test-node-1 uncordoned
   ```

If you need to reboot another node, repeat the procedure.

Moving resources to free up space in a Storage Pool

1. List the Storage Pools on the source node to identify which one is running low on free space:

   ```
   d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list -n
   OLD_NODE
   ```

2. Determine which volumes are located in the overloaded Storage Pool to identify potential candidates for migration:

   ```
   d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor volume list -n
   OLD_NODE
   ```

3. Get the list of resources that own these volumes so that you can proceed with moving them:

   ```
   d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource list-volumes
   ```

4. Create copies of the selected resources on another node (no more than 1–2 at a time):

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-and-i-understand-what-i-am-doing resource create NEW_NODE RESOURCE_NAME
```

5. Wait for the resource synchronization to complete to ensure the data has been replicated properly:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource-definition wait-sync RESOURCE_NAME
```

6. Remove the resource from the source node to free up space in the overloaded Storage Pool:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-and-i-understand-what-i-am-doing resource delete OLD_NODE RESOURCE_NAME
```

Automatic management of replicas and monitoring of LINSTOR state

Replica management and state monitoring are automated in the `replicas_manager.sh` script. It checks the availability of the LINSTOR controller, identifies faulty or corrupted resources, creates database backups, and manages disk replicas, including configuring `TieBreaker` for quorum.

To check the existence of the `replicas_manager.sh` script, run the following command on any master node:

```
ls -l /opt/deckhouse/sbin/replicas_manager.sh
```

Upon execution, the script performs the following actions:

- Verifies the availability of the controller and connectivity to satellites.
- Identifies faulty or corrupted resources.
- Creates a backup of the database.
- Manages the number of disk replicas, adding new ones as needed.
- Configures TieBreaker for resources with two replicas.
- Logs all actions to a file named `linstor_replicas_manager_<date_time>.log`.
- Provides recommendations for resolving issues, such as stuck replicas.

Configuration variables for `replicas_manager.sh`:

- `NON_INTERACTIVE`: Enables non-interactive mode.
- `TIMEOUT_SEC`: Timeout between attempts, in seconds (default: 10).
- `EXCLUDED_RESOURCES_FROM_CHECK`: Regular expression to exclude resources from checks.
- `CHUNK_SIZE`: Chunk size for processing resources (default: 10).
- `NODE_FOR_EVICT`: The name of the node excluded from creating replicas.
- `LINSTOR_NAMESPACE`: Kubernetes namespace (default: `d8-sds-replicated-volume`).
- `DISKLESS_STORAGE_POOL`: Pool for diskless replicas (default: `DfltDisklessStorPool`).

Evicting DRBD resources from a node

Eviction of DRBD resources from a node is performed using the `evict.sh` script. It can operate in two modes:

- Node removal: Additional replicas are created for every resource, after which the node is removed from LINSTOR and Kubernetes.
- Resource removal: Replicas are created for the resources, after which the resources themselves are removed from LINSTOR (the node remains in the cluster).

Preparing and running the script

Before eviction:

1. Make sure the script is present on the master node:

```
ls -l /opt/deckhouse/sbin/evict.sh
```

2. Fix faulty resources:

```
d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor resource list --faulty
```

3. Ensure all Pods in the `d8-sds-replicated-volume` namespace are in the `Running` state:

```
d8 k -n d8-sds-replicated-volume get pods | grep -v Running
```

Example of removing a node from LINSTOR and Kubernetes

Run `evict.sh` on any master node in interactive mode, specifying `--delete-node`:

```
/opt/deckhouse/sbin/evict.sh --delete-node
```

For non-interactive mode, add `--non-interactive` and specify the node name. The script will perform all actions without prompting:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-node --node-name "worker-1"
```

Example of removing resources from a node

Run `evict.sh` on any master node in interactive mode, specifying `--delete-resources-only`:

```
/opt/deckhouse/sbin/evict.sh --delete-resources-only
```

For non-interactive mode, add `--non-interactive` and specify the node name. In this mode, the script runs through all actions without asking for confirmation:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-resources-only --node-name "worker-1"
```

⚠ After the script completes, the node remains in `SchedulingDisabled` status and LINSTOR sets the property `AutoplaceTarget=false`. This blocks automatic creation of new resources on the node.

To allow resource placement again, run:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor'
linstor node set-property "worker-1" AutoplaceTarget
d8 k uncordon "worker-1"
```

Check the `AutoplaceTarget` parameter on all nodes (the field will be empty on nodes where LINSTOR resource placement is allowed):

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor'
linstor node list -s AutoplaceTarget
```

Parameters of the evict.sh script

- `--delete-node`: Remove a node from LINSTOR and Kubernetes after creating additional replicas for all resources on the node.
- `--delete-resources-only`: Remove resources from the node without deleting the node from LINSTOR and Kubernetes, after creating additional replicas.
- `--non-interactive`: Run the script in non-interactive mode.
- `--node-name`: The name of the node to evict resources from. Mandatory when using `--non-interactive`.
- `--skip-db-backup`: Skip creating a LINSTOR DB backup before operations.
- `--ignore-advise`: Perform operations despite `linstor advise resource` warnings. Use if the script was interrupted and the number of replicas for some resources does not match the value

specified in the `ReplicatedStorageClass`.
- `--exclude-resources-from-check`: Exclude resources listed with `|` from checks.

Troubleshooting

Issues can arise at various component layers. The cheat sheet below helps diagnose volume failures in LINSTOR.

Start error of linstor-node while loading the DRBD module

1. Check the `linstor-node` Pods:

   ```
   d8 k get pod -n d8-sds-replicated-volume -l app=linstor-node
   ```

2. If some Pods are stuck in `Init`, check the DRBD version and bashible logs on the node:

   ```
   cat /proc/drbd
   journalctl -fu bashible
   ```

Most likely causes:

- DRBDv8 is loaded instead of the required DRBDv9. Verify the version:

  ```
  cat /proc/drbd
  ```

  If the `/proc/drbd` file is missing, the module is not loaded and the problem lies elsewhere.

- Secure Boot is enabled. Because DRBD is built dynamically (similar to dkms) without a digital signature, the module is not supported when Secure Boot is enabled.

FailedMount error when starting a Pod

Pod stuck in ContainerCreating

If the Pod is stuck in `ContainerCreating` and `d8 k describe pod` shows errors like the one below, the device is mounted on another node:

```
rpc error: code = Internal desc = NodePublishVolume failed for pvc-b3e51b8a-9733-
4d9a-bf34-84e0fee3168d: checking
for exclusive open failed: wrong medium type, check device health
```

Check where the device is in use:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor resource list -r pvc-b3e51b8a-9733-4d9a-bf34-84e0fee3168d
```

The `InUse` flag shows on which node the device is used; unmount the disk manually on that node.

Input/output error

Such errors usually occur during filesystem creation (mkfs). Check `dmesg` on the node where the Pod is starting:

```
dmesg | grep 'Remote failed to finish a request within'
```

If the output contains messages like `Remote failed to finish a request within …`, the disk subsystem may be too slow for DRBD to operate correctly.

After the ReplicatedStoragePool resource is deleted, the corresponding Storage Pool remains in the backend

This is expected behavior. The `sds-replicated-volume` module does not handle deletion operations for the ReplicatedStoragePool resource.

Restrictions on changing ReplicatedStorageClass spec

Only the `isDefault` field can be modified. All other parameters are immutable — this is expected behavior.

Deleting a child StorageClass when deleting a ReplicatedStorageClass

If the StorageClass is in the `Created` status, it can be deleted. For other statuses you must restore the resource or delete the StorageClass manually.

Errors when creating a Storage Pool or StorageClass

For temporary external issues (for example, when `kube-apiserver` is unavailable) the module automatically retries the failed operation.

Error “You’re not allowed to change state of linstor cluster manually”

Operations requiring manual intervention are partially or fully automated in the `sds-replicated-volume` module. Therefore, the module restricts the list of allowed LINSTOR commands. For example, creating a Tie-Breaker is automated because LINSTOR sometimes does not create one for two-replica resources. To see the list of allowed commands, run:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor --help
```

Restoring the database from a backup

Backend resource backups are stored in Secrets as YAML files split into segments for easier restoration. Backups are created automatically on a schedule.

A correctly formed backup looks like this:

```
linstor-20240425074718-backup-0              Opaque                                1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-1              Opaque                                1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-2              Opaque                                1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-completed      Opaque                                0
28s     <none>
```

The backup is stored in encoded segments in Secrets named `linstor-%date_time%-backup-{0..2}`. The Secret `linstor-%date_time%-backup-completed` contains no data and serves as a marker that the backup process completed successfully.

Backup restoration procedure

1. Set environment variables:

   ```
   NAMESPACE="d8-sds-replicated-volume"
   BACKUP_NAME="linstor_db_backup"
   ```

2. List available backups:

   ```
   d8 k -n $NAMESPACE get secrets --show-labels
   ```

   Example output:

```
linstor-20240425072413-backup-0             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-1             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-2             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-completed     Opaque                          0      33m     <none>
linstor-20240425072510-backup-0             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-1             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-2             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-completed     Opaque                          0      32m     <none>
linstor-20240425072634-backup-0             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-1             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-2             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-completed     Opaque                          0      31m     <none>
linstor-20240425072918-backup-0             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-1             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-2             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-completed     Opaque                          0      28m     <none>
linstor-20240425074718-backup-0             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-1             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-2             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-completed     Opaque                          0      10m     <none>
```

3. Each backup has its own creation-time label. Choose the desired one and save it to an environment variable. In this example we use the most recent label:

   ```
   LABEL_SELECTOR="sds-replicated-volume.deckhouse.io/linstor-db-
   backup=20240425074718"
   ```

4. Create a temporary directory to store archive parts:

   ```
   TMPDIR=$(mktemp -d)
   echo "Temporary directory: $TMPDIR"
   ```

5. Create an empty archive and combine Secret data into one file:

   ```
   COMBINED="${BACKUP_NAME}_combined.tar"
   > "$COMBINED"
   ```

6. Get the list of Secrets by label, decode the data, and append it to the archive:

   ```
   MOBJECTS=$(d8 k get rsmb -l "$LABEL_SELECTOR" --sort-by=.metadata.name -o jsonpath="
   {.items[*].metadata.name}")

   for MOBJECT in $MOBJECTS; do
     echo "Process: $MOBJECT"
    d8 k get rsmb "$MOBJECT" -o jsonpath="{.data}" | base64 --decode >> "$COMBINED"
   done
   ```

7. Extract the archive to get backup files:

```
mkdir -p "./backup"
tar -xf "$COMBINED" -C "./backup" --strip-components=2
```

8. Check the backup contents:

   ```
   ls ./backup
   TMPDIR=$(mktemp -d)
   echo "Temporary directory: $TMPDIR"
   ```

   Example output:

   ```
   ebsremotes.yaml                     layerdrbdvolumedefinitions.yaml
   layerwritecachevolumes.yaml  propscontainers.yaml      satellitescapacity.yaml
   secidrolemap.yaml         trackingdate.yaml
   files.yaml                          layerdrbdvolumes.yaml
   linstorremotes.yaml            resourceconnections.yaml  schedules.yaml
   secobjectprotection.yaml  volumeconnections.yaml
   keyvaluestore.yaml                  layerluksvolumes.yaml
   linstorversion.yaml            resourcedefinitions.yaml  secaccesstypes.yaml
   secroles.yaml             volumedefinitions.yaml
   layerbcachevolumes.yaml             layeropenflexresourcedefinitions.yaml
   nodeconnections.yaml           resourcegroups.yaml       secaclmap.yaml
   sectyperules.yaml         volumegroups.yaml
   layercachevolumes.yaml              layeropenflexvolumes.yaml
   nodenetinterfaces.yaml         resources.yaml            secconfiguration.yaml
   sectypes.yaml             volumes.yaml
   layerdrbdresourcedefinitions.yaml  layerresourceids.yaml
   nodes.yaml                     rollback.yaml             secdfltroles.yaml
   spacehistory.yaml
   layerdrbdresources.yaml             layerstoragevolumes.yaml
   nodestorpool.yaml              s3remotes.yaml            secidentities.yaml
   storpooldefinitions.yaml
   ```

9. Restore the required entity by applying the corresponding YAML file:

   ```
   d8 k apply -f %something%.yaml
   ```

   Or bulk-apply for a full restore:

   ```
   d8 k apply -f ./backup/
   ```

Missing sds-replicated-volume service Pods on a selected node

The issue is most likely related to node labels.

- Check `dataNodes.nodeSelector` in module settings:

  ```
  d8 k get mc sds-replicated-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
  ```

- Check the selectors used by `sds-replicated-volume-controller`:

  ```
  d8 k -n d8-sds-replicated-volume get secret d8-sds-replicated-volume-controller-config  -o
  jsonpath='{.data.config}' | base64 --decode
  ```

- The secret `d8-sds-replicated-volume-controller-config` must contain the selectors specified in module settings plus `kubernetes.io/os: linux`.
- Verify that the all labels from the `d8-sds-replicated-volume-controller-config` secret are present on the node:

```
d8 k get node worker-0 --show-labels
```

- If the labels are missing, add them via NodeGroup templates or directly to the node.
- If the labels exist, check whether the node has the label `storage.deckhouse.io/sds-replicated-volume-node=`.
  If not, check whether `sds-replicated-volume-controller` is running and inspect its logs:

```
d8 k -n d8-sds-replicated-volume get po -l app=sds-replicated-volume-controller
d8 k -n d8-sds-replicated-volume logs -l app=sds-replicated-volume-controller
```

Additional support

Reasons for failed operations are shown in the `status.reason` field of ReplicatedStoragePool and ReplicatedStorageClass resources. If that information is insufficient, consult the `sds-replicated-volume-controller` logs.

Migration from the linstor module to sds-replicated-volume

During migration the LINSTOR control plane and its CSI are temporarily unavailable, which can affect PV operations (creation, expansion, or deletion).

⚠ User data is not affected because the migration moves to a new namespace and adds components that manage volumes.

Migration procedure

1. Ensure no faulty resources exist in the backend. The command should return an empty list:

   ```
   alias linstor='d8 k -n d8-linstor exec -ti deploy/linstor-controller -- linstor'
   linstor resource list --faulty
   ```

   i **Warning.** All resources must be healthy before migration.

2. Disable the `linstor` module:

   ```
   d8 k patch moduleconfig linstor --type=merge -p '{"spec": {"enabled": false}}'
   ```

3. Wait until the `d8-linstor` namespace is deleted:

   ```
   d8 k get namespace d8-linstor
   ```

4. Create a ModuleConfig for `sds-node-configurator`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: sds-node-configurator
   spec:
     enabled: true
     version: 1
   EOF
   ```

5. Wait until `sds-node-configurator` reaches `Ready`:

   ```
   d8 k get moduleconfig sds-node-configurator
   ```

6. Create a ModuleConfig for `sds-replicated-volume`:

   > **Warning.** If `settings.dataNodes.nodeSelector` is not specified for `sds-replicated-volume`, its value will be taken from the `linstor` module. If it is absent there as well, it will remain empty and all cluster nodes will be considered data nodes.

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: sds-replicated-volume
   spec:
     enabled: true
     version: 1
   EOF
   ```

7. Wait until `sds-replicated-volume` reaches `Ready`:

   ```
   d8 k get moduleconfig sds-replicated-volume
   ```

8. Check the `sds-replicated-volume` settings:

   ```
   d8 k get moduleconfig sds-replicated-volume -oyaml
   ```

9. Wait until all Pods in the `d8-sds-replicated-volume` and `d8-sds-node-configurator` namespaces are `Ready` or `Completed`:

   ```
   d8 k get po -n d8-sds-node-configurator
   d8 k get po -n d8-sds-replicated-volume
   ```

10. Update the `linstor` alias and check resources:

    ```
    alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor'
    linstor resource list --faulty
    ```

If no faulty resources are found, migration was successful.

Migrating to ReplicatedStorageClass

StorageClasses in this module are managed via the ReplicatedStorageClass resource. StorageClasses must not be created manually.

When migrating from the LINSTOR module, delete old StorageClasses and create new ones via ReplicatedStorageClass according to the table below.

Note that in old StorageClasses you look at the option in the `parameters` section of the StorageClass itself, while when creating a new one you set the corresponding option in ReplicatedStorageClass.

StorageClass parameter	ReplicatedStorageClass	Default	Notes
linstor.csi.linbit.com/placementCount: “1”	replication: “None”		One data replica will be created
linstor.csi.linbit.com/placementCount: “2”	replication: “Availability”		Two data replicas will be created
linstor.csi.linbit.com/placementCount: “3”	replication: “ConsistencyAndAvailability”	Yes	Three data replicas will be created
linstor.csi.linbit.com/storagePool: “name”	storagePool: “name”		Name of the storage pool used for storage

StorageClass parameter	ReplicatedStorageClass	Default	Notes
linstor.csi.linbit.com/allowRemoteVolumeAccess: “false”	volumeAccess: “Local”		Remote Pod access to data volumes is forbidden (local disk access within the node only)

Additional parameters:

- `reclaimPolicy` (Delete, Retain): Corresponds to `reclaimPolicy` of the old StorageClass.
- `zones`: List of zones to place resources in (direct cloud zone names). Note that remote Pod access to the data volume is possible only within one zone.
- `volumeAccess` values: `Local` (access strictly within the node), `EventuallyLocal` (a data replica will synchronize to the node after the Pod starts), `PreferablyLocal` (remote Pod access allowed, `volumeBindingMode: WaitForFirstConsumer`), `Any` (remote Pod access allowed, `volumeBindingMode: Immediate`).
- If you need `volumeBindingMode: Immediate`, set `volumeAccess` in ReplicatedStorageClass to `Any`.

Migrating to ReplicatedStoragePool

The ReplicatedStoragePool resource allows you to create Storage Pools in the backend. It is recommended to create this resource even for existing Storage Pools and reference existing LVMVolumeGroup. The controller will detect that the Storage Pools already exist and leave them unchanged, showing `Created` in `status.phase`.

Migration from the sds-drbd module to sds-replicated-volume

During migration the module control plane and its CSI are unavailable. This prevents creation, expansion, or deletion of PVs and the creation or deletion of Pods that use DRBD PVs for the duration of the migration.

⚠ The migration will not affect user data, as it is performed in a new namespace and volume management will be handled by new components that will replace the functionality of the previous module.

Migration procedure

1. Make sure there are no faulty DRBD resources in the cluster. The command should return an empty list:

   ```
   alias linstor='d8 k -n d8-sds-drbd exec -ti deploy/linstor-controller -- linstor'
   linstor resource list --faulty
   ```

   Warning. All DRBD resources must be healthy before migration.

2. Disable the `sds-drbd` module:

   ```
   d8 k patch moduleconfig sds-drbd --type=merge -p '{"spec": {"enabled": false}}'
   ```

3. Wait until the `d8-sds-drbd` namespace is deleted:

   ```
   d8 k get namespace d8-sds-drbd
   ```

4. Create a ModuleConfig for `sds-replicated-volume`:

   Warning. If `settings.dataNodes.nodeSelector` is not specified for `sds-replicated-volume`, its value will be taken from the `sds-drbd` module. If it is absent there as well, it will remain empty and all cluster nodes will be considered data nodes.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

5. Wait until `sds-replicated-volume` reaches `Ready`:

```
d8 k get moduleconfig sds-replicated-volume
```

6. Check the `sds-replicated-volume` settings:

```
d8 k get moduleconfig sds-replicated-volume -oyaml
```

7. Wait until all Pods in the `d8-sds-replicated-volume` namespace are `Ready` or `Completed`:

```
d8 k get po -n d8-sds-replicated-volume
```

8. Update the `linstor` alias and check DRBD resources:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-
controller -- linstor'
linstor resource list --faulty
```

If no faulty resources are found, migration was successful.

> **Warning.** DRBDStoragePool and DRBDStorageClass resources will be automatically migrated to ReplicatedStoragePool and ReplicatedStorageClass. No user action is required.

The logic of these resources remains unchanged. However, verify that no DRBDStoragePool or DRBDStorageClass resources remain. If they do, contact the Deckhouse technical support.

Reasons to avoid RAID with sds-replicated-volume

Using DRBD with more than one replica already provides network-level RAID functionality. Local RAID can cause the following issues:

- Significantly increases space overhead when using redundant RAID.
  Example: ReplicatedStorageClass with `replication` set to `ConsistencyAndAvailability`. DRBD will store three data replicas (one per node). If those nodes use RAID1, storing 1 GB of data will require 6 GB of disk space. Redundant RAID is reasonable only to simplify server maintenance when storage cost is irrelevant. RAID1 then allows replacing disks without moving data replicas off a “problem” disk.
- With RAID0 the performance gain is negligible because data replication occurs over the network and the bottleneck is likely the network. Additionally, reduced host storage reliability can lead to data unavailability since DRBD failover from a broken replica to a healthy one is not instantaneous.

Recommendations for using local disks

DRBD uses the network for data replication. When using NAS, network load increases dramatically because nodes synchronize data not only with the NAS but also with each other. Latency for reads or writes also increases. NAS typically uses RAID on its side, adding further overhead.

Manual trigger the certificate renewal process

Although the certificate renewal process is automated, manual renewal might still be necessary because it can be performed during a convenient maintenance window when it is acceptable to restart

the module's objects. The automated renewal does not restart any objects.

To manually trigger the certificate renewal process, create a `ConfigMap` named `manualcertrenewal-trigger`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manualcertrenewal-trigger
  namespace: d8-sds-replicated-volume
```

The system will stop all necessary module objects, update the certificates, and then restart them.

You can check the operation status using the following command:

```
d8 k -n d8-sds-replicated-volume get cm manualcertrenewal-trigger -ojsonpath='{.data.step}'
```

Possible statuses:

- `Prepared`: Health checks have passed successfully, and the downtime window has started.
- `TurnedOffAndRenewedCerts`: The system has been stopped and certificates have been renewed.
- `TurnedOn`: The system has been restarted.
- `Done`: The operation is complete and ready to be repeated.

Certificates are issued for a period of one year and are marked as expiring 30 days before their expiration date. The monitoring system alerts about expiring certificates using the `D8LinstorCertificateExpiringIn30d` alert.

To repeat the operation, simply remove the label from the trigger using the following command:

```
d8 k -n d8-sds-replicated-volume label cm manualcertrenewal-trigger storage.deckhouse.io/sds-replicated-volume-manualcertrenewal-completed-
```

Additional resources

- Module `sds-local-volume` parameters
- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Distributed Ceph storage

Ceph is a scalable distributed storage system that ensures high availability and fault tolerance of data. Deckhouse Platform (DP) provides Ceph cluster integration using the `csi-ceph` module. This enables dynamic storage management and the use of StorageClass based on RADOS Block Device (RBD) or CephFS.

> The snapshot-controller module is required for working with snapshots.

This page provides instructions on connecting Ceph to Deckhouse, configuring authentication, creating StorageClass objects, and verifying storage functionality.

Migration from `ceph-csi` module

When switching from the `ceph-csi` module to `csi-ceph`, an automatic migration is performed, but its execution requires preliminary preparation:

1. Set the replica count to zero for all operators (redis, clickhouse, kafka, etc.). Exception: the `prometheus` operator will be disabled automatically.

2. Disable the `ceph-csi` module and enable `csi-ceph`.
3. Wait for the operation to complete. The Deckhouse logs should show the message “Finished migration from Ceph CSI module”.
4. Verify functionality. Create test pods and PVCs to test CSI.
5. Restore the operators to their normal replica count.

⚠ If Ceph StorageClass was created without using the `CephCSIDriver` resource, manual migration will be required. Contact technical support.

Connecting to a Ceph cluster

To connect to a Ceph cluster, follow the step-by-step instructions below. Execute all commands on a machine with administrative access to the Kubernetes API.

1. Execute the command to activate the `csi-ceph` module:

   ```
   d8 s module enable csi-ceph
   ```

2. Wait for the module to transition to `Ready` state:

   ```
   d8 k get module csi-ceph -w
   ```

3. Ensure that all pods in the `d8-csi-ceph` namespace are in `Running` or `Completed` state and deployed on all cluster nodes:

   ```
   d8 k -n d8-csi-ceph get pod -owide -w
   ```

4. To configure the connection to the Ceph cluster, apply the CephClusterConnection resource.

   Example command:

   ```
   d8 k apply -f - <<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: CephClusterConnection
   metadata:
     name: ceph-cluster-1
   spec:
     # FSID/UUID of the Ceph cluster.
     # Get the FSID/UUID of the Ceph cluster using the command `ceph fsid`.
     clusterID: 2bf085fc-5119-404f-bb19-820ca6a1b07e
     # List of IP addresses of ceph-mon in the format 10.0.0.10:6789.
     monitors:
       - 10.0.0.10:6789
     # Username without `client.`.
     # Get the username using the command `ceph auth list`.
     userID: admin
     # Authorization key corresponding to userID.
     # Get the authorization key using the command `ceph auth get-key
   client.admin`.
     userKey: AQDiVXVmBJVRLxAAg65PhODrtwbwSWrjJwssUg==
   EOF
   ```

5. Verify the connection creation with the command (`Phase` should be in `Created` status):

   ```
   d8 k get cephclusterconnection ceph-cluster-1
   ```

6. Create a StorageClass object using the CephStorageClass resource. Manual creation of StorageClass without using CephStorageClass may lead to errors.

   Example of creating StorageClass based on RBD:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-rbd-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: RBD
  rbd:
    defaultFSType: ext4
    pool: ceph-rbd-pool
EOF
```

Example of creating StorageClass based on Ceph filesystem:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-fs-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: CephFS
  cephFS:
    fsName: cephfs
EOF
```

7. Verify that the created CephStorageClass resources have transitioned to `Created` state:

   ```
   d8 k get cephstorageclass
   ```

   This will output information about the created CephStorageClass resources:

   ```
   NAME          PHASE     AGE
   ceph-rbd-sc   Created   1h
   ceph-fs-sc    Created   1h
   ```

8. Verify the created StorageClass:

   ```
   d8 k get sc
   ```

   This will output information about the created StorageClass:

   ```
   NAME          PROVISIONER        RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   ceph-rbd-sc   rbd.csi.ceph.com   Delete          WaitForFirstConsumer   true
   15s
   ceph-fs-sc    rbd.csi.ceph.com   Delete          WaitForFirstConsumer   true
   15s
   ```

Ceph cluster connection setup is complete. You can use the created StorageClass to create PersistentVolumeClaim in your applications.

Additional information

How to get a list of RBD volumes separated by nodes

For monitoring and diagnostics, it’s useful to know which RBD volumes are connected to each cluster node. The following command provides detailed information about volume mapping:

```
d8 k -n d8-csi-ceph get po -l app=csi-node-rbd -o custom-
columns=NAME:.metadata.name,NODE:.spec.nodeName --no-headers \
  | awk '{print "echo "$2"; kubectl -n d8-csi-ceph exec  "$1" -c node -- rbd
showmapped"}' | bash
```

Which versions of Ceph clusters are supported

The `csi-ceph` module has specific requirements for the Ceph cluster version to ensure compatibility and stable operation. Officially supported versions are >= 16.2.0. In practice, the current version works with clusters running versions >= 14.2.0, but it’s recommended to update Ceph to the latest version.

Which volume access modes are supported

Different types of Ceph storage support different volume access modes, which is important to consider when planning application architecture.

- **RBD**: Supports only ReadWriteOnce (RWO) — access to volume from only one cluster node.
- **CephFS**: Supports ReadWriteOnce (RWO) and ReadWriteMany (RWX) — simultaneous access to volume from multiple cluster nodes.

Examples of permissions (caps) for Ceph users

To ensure proper operation of the `csi-ceph` module, Ceph users must have appropriate permissions (caps) configured. The required permissions depend on the storage type being used (RBD, CephFS, or both). Below are examples of correct permission configurations for different scenarios.

RBD

For a single pool named `rbd`, the following permissions are required:

```
[client.name]
        key = key
        caps mgr = "profile rbd pool=rbd"
        caps mon = "profile rbd"
        caps osd = "profile rbd pool=rbd"
```

CephFS

Before configuring CephFS permissions, ensure that a subvolumegroup `csi` (or another one specified in `Custom resources`) is created in CephFS.

You can create a new subvolumegroup using the following command on the Ceph management node:

```
ceph fs subvolumegroup create <fs_name> <group_name>
```

For example, to create a subvolumegroup `csi` for filesystem `myfs`:

```
ceph fs subvolumegroup create myfs csi
```

Required permissions for CephFS named `myfs`:

```
[client.name]
        key = key
        caps mds = "allow rwps fsname=myfs"
        caps mgr = "allow rw"
        caps mon = "allow r fsname=myfs"
        caps osd = "allow rw tag cephfs data=myfs, allow rw tag cephfs
metadata=myfs"
```

CephFS + RBD

For a user that needs access to both CephFS `myfs` and RBD pool `rbd`, combine the permissions as follows:

```
[client.name]
        key = key
        caps mds = "allow rwps fsname=myfs"
        caps mgr = "allow rw,profile rbd pool=rbd"
        caps mon = "allow r fsname=myfs,profile rbd"
        caps osd = "allow rw tag cephfs metadata=myfs, allow rw tag cephfs
data=myfs,profile rbd pool=rbd"
```

Additional resources

- Module `csi-ceph` custom resources
- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

HPE data storage

Deckhouse includes support for HPE 3PAR storage systems, enabling volume management in Kubernetes using a CSI driver. This integration provides reliable, scalable, and high-performance storage suitable for mission-critical workloads. The `csi-hpe` module is used to work with HPE 3PAR systems, allowing StorageClass creation in Kubernetes through the HPEStorageClass resource.

> ⚠ User-created StorageClass for the `csi.hpe.com` CSI driver is not allowed.
> Only HPE 3PAR storage systems are supported. For other HPE storage systems, contact the Deckhouse technical support.

This page provides instructions on connecting HPE 3PAR to Deckhouse, configuring the connection, creating StorageClass, and verifying storage functionality.

System requirements

- A deployed and configured HPE storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuration

Note that all commands must be run on a machine that has administrator access to the Kubernetes API.

Enabling the module

Enable the `csi-hpe` module. This will result in the following actions across all cluster nodes:

- Registration of the CSI driver.
- Launch of service Pods for the `csi-hpe` components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-hpe
spec:
  enabled: true
  version: 1
EOF
```

Wait until the module transitions to the `Ready` state. Check the module's status with the following command:

```
d8 k get module csi-hpe -w
```

Creating a StorageClass

To create a StorageClass, you need to use the HPEStorageClass and HPEStorageConnection resource. Here is an example command to create such a resource:

- Creating a HPEStorageConnection resource:

  ```
  d8 k apply -f -<<EOF
  apiVersion: storage.deckhouse.io/v1alpha1
  kind: HPEStorageConnection
  metadata:
    name: hpe
  spec:
    controlPlane:
      backendAddress: "172.17.1.55" # Storage system address (mutable).
      username: "3paradm" # API username (mutable).
      password: "3pardata" # API password (mutable).
      serviceName: "primera3par-csp-svc"
      servicePort: "8080"
  EOF
  ```

  Verify the creation of the object using the following command (`Phase` should be `Created`):

  ```
  d8 k get hpestorageconnections.storage.deckhouse.io <hpestorageconnection-name>
  ```

- Creating a HPEStorageClass resource:

  ```
  d8 k apply -f -<<EOF
  apiVersion: storage.deckhouse.io/v1alpha1
  kind: HPEStorageClass
  metadata:
    name: hpe
  spec:
    pool: "test-cpg"
    accessProtocol: "iscsi" # fc or iscsi (iscsi by default), immutable.
    fsType: "xfs" # xfs, ext3, ext4, btrfs (ext4 by default), mutable.
    storageConnectionName: "hpe" # Immutable.
    reclaimPolicy: Delete # Delete or Retain.
    cpg: "test-cpg"
  EOF
  ```

  Verify the creation of the object using the following command (`Phase` should be `Created`):

  ```
  d8 k get hpestorageclasses.storage.deckhouse.io <hpestorageclass-name>
  ```

Module health verification

To verify module health, ensure that all Pods in the `d8-csi-hpe` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-hpe get pod -owide -w
```

Additional resources

- Module `csi-hpe` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Huawei data storage

Deckhouse provides support for Huawei Dorado storage systems, enabling volume management in Kubernetes using a CSI driver through the creation of custom resources like HuaweiStorageClass. This solution ensures high-performance and fault-tolerant storage, making it an optimal choice for mission-critical workloads.

> ⚠ User-created StorageClass for the `csi.huawei.com` CSI driver is not allowed.
> Only Huawei Dorado storage systems are supported. For other Huawei storage systems, contact the Deckhouse technical support.

This page provides instructions on connecting Huawei Dorado to Deckhouse, configuring the connection, creating StorageClass, and verifying storage functionality.

System requirements

- Presence of a deployed and configured Huawei storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuration

Note that all commands must be run on a machine that has administrator access to the Kubernetes API.

Enabling the module

To support Huawei Dorado storage systems, enable the `csi-huawei` module. This will ensure that all cluster nodes have:

- Registration of the CSI driver.
- Launch of service pods for the `csi-huawei` components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-huawei
spec:
  enabled: true
  version: 1
EOF
```

Wait until the module transitions to the `Ready` state. Check the module's status with the following command:

```
d8 k get module csi-huawei -w
```

Creating a StorageClass

To create a StorageClass, you need to use the HuaweiStorageClass and HuaweiStorageConnection resource. Here is an example command to create such a resource:

- Creating a HuaweiStorageConnection resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageConnection
metadata:
  name: huaweistorageconn
spec:
  storageType: OceanStorSAN
  pools:
    - test
  urls:
    - https://192.168.128.101:8088
  login: "admin"
  password: "dh-test-secret-0123456789abcdef"
  protocol: ISCSI
  portals:
    - 10.240.0.101
    - 10.250.0.101
  maxClientThreads: 30

EOF
```

Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get huaweistorageconnections.storage.deckhouse.io <huaweistorageconnection name>
```

- Creating a HuaweiStorageClass resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageClass
metadata:
  name: huaweisc
spec:
  fsType: ext4
  pool: test
  reclaimPolicy: Delete
  storageConnectionName: huaweistorageconn
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get huaweistorageclasses.storage.deckhouse.io <huaweistorageclass name>
```

Module health verification

To verify module health, ensure that all pods in the `d8-csi-huawei` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-huawei get pod -owide -w
```

Additional resources

- Module `csi-huawei` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

NetApp data storage

Deckhouse Platform (DP) implements support for NetApp data storage systems for volume management in Kubernetes using CSI driver. This integration provides reliable, scalable, and high-performance storage suitable for critical workloads. For working with NetApp storage systems, the `csi-netapp` module is used, which allows creating StorageClass in Kubernetes through creating a NetappStorageClass resource.

> ⚠ Creating StorageClass for `csi.netapp.com` CSI driver by users is prohibited. Currently, the module supports storage systems compatible with NetApp Trident CSI. For support of other NetApp storage systems, please contact Deckhouse technical support.

This page provides instructions for connecting NetApp to DP, configuring the connection, and creating StorageClass.

System Requirements

Before configuring work with the NetApp storage system, make sure that the following requirements are met:

- Availability of deployed and configured NetApp storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuring cluster integration with the NetApp storage system

To start working with the NetApp storage system, follow the step-by-step instructions below. Run all commands on a machine with administrative access to the Kubernetes API.

> To work with snapshots, the snapshot-controller module must be connected.

1. Execute the command to activate the `csi-netapp` module.

   ```
   d8 system module enable csi-netapp
   ```

   After activation, the following will be deployed on all cluster nodes:

   - CSI driver will be registered.
   - Service pods of `csi-netapp` components will be deployed.

2. Wait for the module to transition to the `Ready` state:

   ```
   d8 k get module csi-netapp -w
   ```

3. Ensure that all pods in the `d8-csi-netapp` namespace are in the `Running` or `Completed` state and deployed on all cluster nodes:

   ```
   d8 k -n d8-csi-netapp get pod -owide -w
   ```

4. Create a NetappStorageConnection resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NetappStorageConnection
metadata:
  name: netapp
spec:
  controlPlane:
    address: "172.17.1.55"
    username: "admin"
    password: "password"
    svm: "svm1"
EOF
```

5. Verify object creation with the following command (`Phase` should be `Created`):

```
d8 k get netappstorageconnections.storage.deckhouse.io <netappstorageconnection name>
```

6. Create a NetappStorageClass resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NetappStorageClass
metadata:
  name: netapp
spec:
  pool: "test-cpg"
  accessProtocol: "iscsi" # fc or iscsi (default iscsi), immutable parameter.
  fsType: "xfs" # xfs, ext3, ext4 (default ext4), configurable parameter.
  storageConnectionName: "netapp" # Immutable parameter.
  reclaimPolicy: Delete # Delete or Retain.
  cpg: "test-cpg"
EOF
```

7. Verify object creation with the following command (`Phase` should be `Created`):

```
d8 k get netappstorageclasses.storage.deckhouse.io <netappstorageclass name>
```

The NetApp storage system is now ready for use. You can use the created StorageClass to create PersistentVolumeClaims in your applications.

Additional resources

- Module `csi-netapp` documentation
- Module `snapshot-controller` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

NFS storage

Deckhouse Platform (DP) supports integration with Network File System (NFS), providing the ability to use network file storage as Kubernetes volumes. The `csi-nfs` module provides a CSI driver for connecting to NFS servers and creating PersistentVolumes based on them.

This page provides instructions for configuring NFS storage in DP, including connecting to an NFS server, creating StorageClass, configuring RPC-with-TLS security, and verifying system functionality.

System Requirements

Minimum Requirements

The following conditions must be met for the `csi-nfs` module to work:

- Supported Linux distributions with appropriate kernels.
- Configured and accessible NFS server.
- For RPC-with-TLS support: Linux kernel with enabled `CONFIG_TLS` and `CONFIG_NET_HANDSHAKE` options.

Configuration Recommendations

For optimal module operation, it is recommended to:

- For automatic pod restarts when TLS parameters change, enable the `pod-reloader` module (enabled by default).
- Use stable versions of NFS servers with support for required protocols.

⚠ For NFS to work as virtual disk storage in Deckhouse Virtualization Platform, configure the NFS server with the `no_root_squash` option.

Limitations

General Limitations

The following limitations apply when working with the `csi-nfs` module:

- Manual creation of StorageClass for CSI driver `nfs.csi.k8s.io` is prohibited — use the NFSStorageClass resource.
- PersistentVolumes are created only through NFSStorageClass resources.
- Changing NFS server parameters in already created PVs is impossible.

Volume Snapshot Limitations

i A connected `snapshot-controller` module is required for working with snapshots.

Creating NFS volume snapshots has significant limitations related to the NFS architecture and the way they are implemented in `csi-nfs`. Avoid using snapshots in module whenever possible.

Snapshot Operation Principle:

- The CSI driver creates a snapshot at the NFS server level.
- The `tar` utility is used for archiving, which imposes certain limitations.
- The archive is saved in the root folder of the NFS server specified in the `spec.connection.share` parameter.

⚠
- Before creating a snapshot, **mandatory** stop the workload (pods) using the NFS volume.
- NFS does not ensure atomicity of operations at the file system level when creating snapshots.

RPC-with-TLS Mode Limitations

Functional Limitations

The following limitations apply when using RPC-with-TLS:

- Only one client certificate is supported for the `mtls` security policy.
- One NFS server cannot simultaneously work in different security modes: `tls`, `mtls` and standard mode (without TLS).
- The `tlshd` daemon should not be running on cluster nodes, otherwise it will conflict with the module daemon. To prevent conflicts when enabling TLS on nodes, third-party `tlshd` is automatically stopped and its autostart is disabled.

System Requirements

The following system requirements must be met for RPC-with-TLS operation:

- Linux kernel must be compiled with enabled `CONFIG_TLS` and `CONFIG_NET_HANDSHAKE` parameters.
- `nfs-utils` package (in Debian-based distributions — `nfs-common`) version >= 2.6.3.

Configuration

> **i** All commands must be executed on a machine with administrative rights in the Kubernetes API.

The following steps are required to configure NFS storage:

- Module enabling.
- Creating NFSStorageClass.

Enabling the Module

To support working with NFS storage, enable the `csi-nfs` module, which allows creating StorageClass in Kubernetes using custom NFSStorageClass resources. After enabling the module, the following will happen on cluster nodes:

- CSI driver registration.
- Launch of `csi-nfs` service pods and creation of necessary components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-nfs
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state:

```
d8 k get module csi-nfs -w
```

Check the status of pods in the `d8-csi-nfs` namespace. All pods should be in `Running` or `Completed` state and running on all nodes:

```
d8 k -n d8-csi-nfs get pod -owide -w
```

Creating StorageClass

To create a StorageClass, you must use the NFSStorageClass resource. Example of creating a resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NFSStorageClass
metadata:
  name: nfs-storage-class
spec:
  connection:
    host: 10.223.187.3
    share: /
    nfsVersion: "4.1"
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

For each PV, a directory `<share directory>/<PV name>` will be created.

Volume Cleanup Configuration

When deleting a PersistentVolume, files with user data may remain on the volume. To ensure data security, you can configure the volume cleanup method before deletion using the `volumeCleanup` parameter.

⚠ Important notes on volume cleanup:

- The cleanup option does not affect files already deleted by the client application.
- Cleanup is performed only through the NFS protocol and depends on:
  - NFS server service.
  - File system.
  - Block device level and their virtualization (e.g., LVM).
  - Physical devices.
- Ensure NFS server trustworthiness before sending sensitive data.

Parameter `RandomFillSinglePass`

File contents are overwritten with a random sequence before deletion. The random sequence is transmitted over the network.

Parameter `RandomFillThreePass`

File contents are overwritten three times with random sequences before deletion. Three random sequences are transmitted over the network.

i Using this method makes sense only if the server stores data on a hard disk, and there is a risk of physical access by an attacker to the device.

Parameter `Discard`

An optimized cleanup method that uses file system capabilities for working with solid-state drives. File contents are marked as free through the `falloc` system call with the `FALLOC_FL_PUNCH_HOLE` flag. The file system will free blocks fully used by the file through the `blkdiscard` call, and the remaining space will be overwritten with zeros.

Advantages of the `Discard` method:

- Traffic volume does not depend on file size, only on their quantity.
- Can ensure inaccessibility of old data in some server configurations.
- Works for both hard drives and solid-state drives.
- Allows increasing solid-state drive lifespan.

Changing NFS Server Parameters for Created PersistentVolumes

Changing NFS server parameters for created PersistentVolumes is impossible, as connection data to the NFS server is stored directly in the PersistentVolume manifest and cannot be changed. Changing NFSStorageClass will also not affect connection settings in existing PVs.

Creating Volume Snapshots

In `csi-nfs`, snapshots are created by archiving the volume folder. The archive is saved in the root of the NFS server folder specified in the `spec.connection.share` parameter. When creating volume snapshots, it’s important to consider the requirements and limitations listed above.

To create a snapshot, perform the following steps:

1. Enable the `snapshot-controller` module:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: snapshot-controller
spec:
  enabled: true
  version: 1
EOF
```

2. Create a volume snapshot, specifying the required parameters:

```
d8 k apply -f - <<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: my-snapshot
  namespace: <namespace name where the PVC is located>
spec:
  volumeSnapshotClassName: csi-nfs-snapshot-class
  source:
    persistentVolumeClaimName: <PVC name for which the snapshot needs to be
created>
EOF
```

3. Check the status of the created snapshot:

```
d8 k get volumesnapshot
```

Issues with Deleting PVs with RPC-with-TLS Support

If the NFSStorageClass resource was configured with RPC-with-TLS support, a situation may arise where the PersistentVolume cannot be deleted. This happens due to secret deletion (e.g., after deleting NFSStorageClass) that stores mounting parameters. As a result, the controller cannot mount the NFS folder to delete the `<PV name>` folder.

Adding Multiple CAs to the `tlsParameters.ca` Parameter

To add multiple CA certificates to the `tlsParameters.ca` parameter, use the following commands:

For two CAs:

```
cat CA1.crt CA2.crt | base64 -w0
```

For three CAs:

```
cat CA1.crt CA2.crt CA3.crt | base64 -w0
```

For more CAs:

```
cat CA1.crt CA2.crt CA3.crt ... CAN.crt | base64 -w0
```

Additional resources

- Module `csi-nfs` documentation
- Module `pod-reloader` documentation
- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

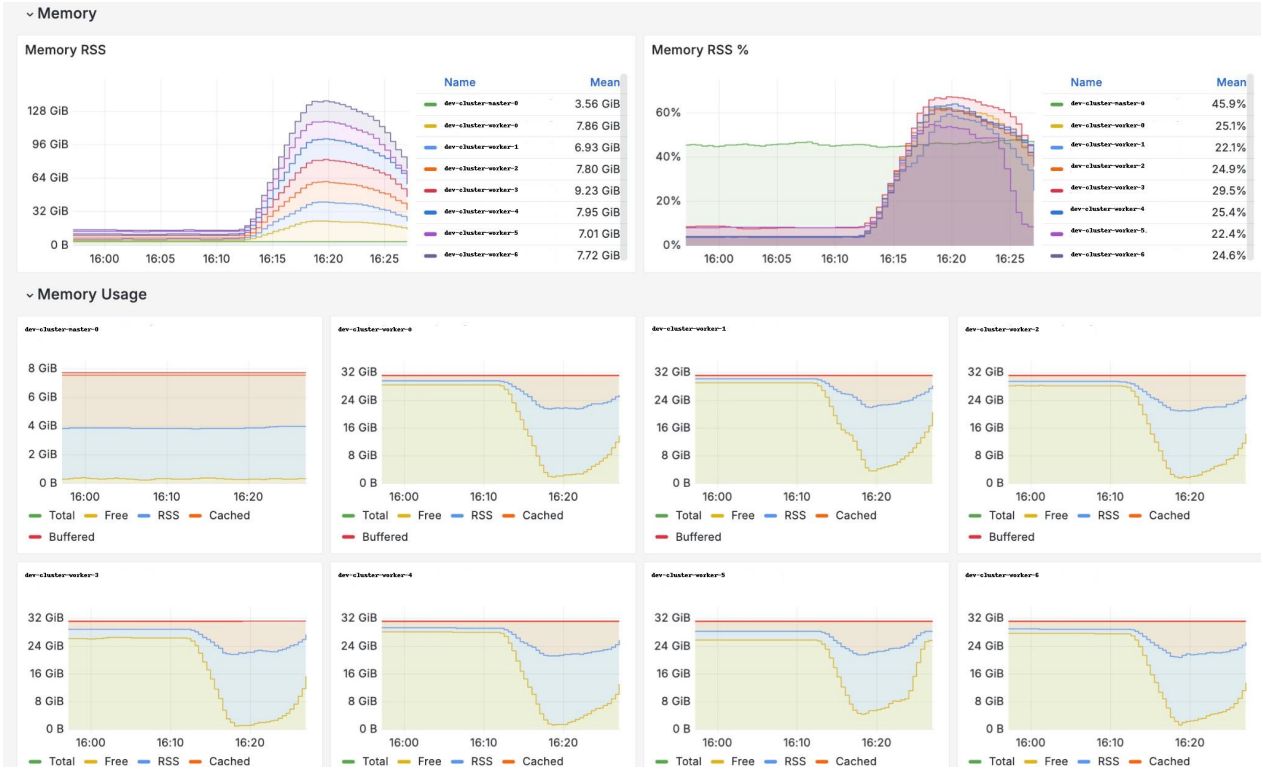
S3 object storage

Deckhouse supports working with S3-based object storage, enabling its use in Kubernetes for storing data as volumes. The GeeseFS file system is used, running via FUSE on top of S3, allowing S3 storage to be mounted as standard file systems.

This page provides instructions for setting up S3 storage in Deckhouse, including connection, creating a StorageClass, and verifying system functionality.

System requirements

- Kubernetes version 1.17+ with support for privileged containers.
- A configured S3 storage with available access keys.
- Sufficient memory on nodes. `GeeseFS` uses caching for working with files retrieved from S3. The cache size is set using the `maxCacheSize` parameter in S3StorageClass.
  Stress test results: 7 nodes, 600 Pods and PVCs, `maxCacheSize` = 500 MB, each Pod writes 300 MB, reads it, and then terminates.



Configuration

Note that all commands must be executed on a machine with administrative privileges in the Kubernetes API.

Required steps:

- Enable the module.
- Create S3StorageClass.

Enabling the module

To support working with S3 storage, enable the `csi-s3` module, which allows the creation of StorageClass and Secret in Kubernetes using custom resources like S3StorageClass.
After enabling the module, the following will occur on cluster nodes:

- CSI driver registration.

- Launch of `csi-s3` service Pods and creation of necessary components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-s3
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state.

```
d8 k get module csi-s3 -w
```

Creating a StorageClass

The module is configured via the S3StorageClass manifest. Below is an example configuration:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: S3StorageClass
metadata:
  name: example-s3
spec:
  bucketName: example-bucket
  endpoint: https://s3.example.com
  region: us-east-1
  accessKey: <your-access-key>
  secretKey: <your-secret-key>
  maxCacheSize: 500
  insecure: false
```

If `bucketName` is empty then bucket in S3 will be created for each PV. If `bucketName` is not empty then folder inside the bucket will be created for each PV. If the specifies bucket does not exist it will be created.

Module health check

To verify the health of the module, ensure that all Pods in the `d8-csi-s3` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-s3 get pod -owide -w
```

Limitations

S3 is not a traditional file system, so it comes with several limitations. POSIX compatibility depends on the mounting module used and the specific S3 provider. Some storage backends may not guarantee data consistency (more details here).

Pay attention to the POSIX compatibility matrix.

Key limitations:

- File permissions, symbolic links, user-defined `mtimes`, and special files (block/character devices, named pipes, UNIX sockets) are not supported.
- Special file support is enabled by default for `Yandex S3` but disabled for other providers.
- File permissions are disabled by default.
- User-defined modification times are also disabled: `ctime`, `atime`, and `mtime` are always the same.
- The file modification time cannot be set manually (e.g., using `cp --preserve`, `rsync -a`, or `utimes(2)`).
- Hard links are not supported.
- File locking is not supported.
- “Invisible” deleted files are not retained. If an application keeps an open file descriptor after

deleting a file, it will receive `ENOENT` errors when trying to access it.
- The default file size limit is 1.03 TB, achieved through chunking: 1000 parts of 5 MB, 1000 parts of 25 MB, and 8000 parts of 125 MB. The chunk size can be adjusted, but AWS enforces a maximum file size of 5 TB.

Known bugs

- The requested PVC volume size does not affect the created S3 bucket.
- `df -h` always reports the mounted storage size as 1 PB, and `used` does not change during usage.
- The CSI driver does not validate storage access credentials. Even with incorrect keys, the Pod will remain in `Running` status, and PersistentVolume and PersistentVolumeClaim will be `Bound`. Any attempt to access the mounted directory within the Pod will result in the Pod restarting.

Changing connection parameters for existing PVs

Connection parameters for storage cannot be changed. Changing the StorageClass does not update connection settings in existing PVs.

Displaying the mounted directory size

The `df -h` command shows the mounted volume size as 1 PB, and the `Used` value remains unchanged during operation. This is a peculiarity of the `GeeseFS` mounting module.

Handling exceeded bucket or user quotas

Exceeding the quota is an abnormal situation that should be avoided. The behavior in this case depends on the backend:

- Files can be copied and edited in Pods, but changes will not be reflected in the storage.
- The Pod may fail and restart.

Obtaining information about used space

To get information about the used space, use the storage web interface or a command-line utility.

Using multiple S3 storages in a single module

To do this, create an additional S3StorageClass and a corresponding PVC, then mount them in a Pod as follows:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: csi-s3-test-nginx
  namespace: default
spec:
  containers:
    - name: csi-s3-test-nginx
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html/s3
          name: webroot
        - mountPath: /opt/homedir
          name: homedir
  volumes:
    - name: webroot
      persistentVolumeClaim:
        claimName: csi-s3-pvc # PVC name
        readOnly: false
    - name: homedir
      persistentVolumeClaim:
        claimName: csi-s3-pvc2 # PVC-2 name
        readOnly: false
  EOF
```

Using a single bucket for multiple Pods

When the `bucketName` parameter is specified in S3StorageClass, separate directories are created for each PV in the same bucket, allowing one bucket to be used for multiple Pods.

Troubleshooting

Issues creating a PVC

If you encounter problems when creating a PVC, check the `provisioner` logs:

```
d8 k -n d8-csi-s3 logs -l app=csi-provisioner-s3 -c csi-s3
```

Issues starting containers

Make sure that MountPropagation is not set to `false`, and check the S3 driver logs:

```
d8 k -n d8-csi-s3 logs -l app=csi-s3 -c csi-s3
```

Additional resources

- Module `csi-s3` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

SCSI-based data storage

Deckhouse Platform (DP) supports managing storage connected via iSCSI or Fibre Channel, enabling working with volumes at the block device level. This allows for the integration of storage systems with Kubernetes and management through a CSI driver.

This page provides instructions for connecting SCSI devices in DP, creating SCSITarget, StorageClass, and verifying system functionality.

Supported features

DP supports:

- Detecting Logical Unit Numbers (LUN) via iSCSI or Fibre Channel.
- Creating PersistentVolume (PV) from pre-provisioned LUN.
- Deleting PersistentVolume and wiping data on LUN.
- Attaching LUN to nodes via iSCSI or Fibre Channel.
- Creating multipath devices and mounting them in pods.
- Detaching LUN from nodes.

Limitations

DP doesn’t support:

- Creating LUN on storage systems.
- Resizing LUN.
- Creating snapshots.

System requirements

The following requirements are applicable to the infrastructure and cluster nodes:

- A deployed and configured storage system providing access to LUN via iSCSI or Fibre Channel.
- For iSCSI connections:
  - A unique iSCSI Qualified Name (IQN) must be configured on each cluster node in the

    - `/etc/iscsi/initiatorname.iscsi` file.
    - The `multipath-tools` package must be installed on nodes.
- For Fibre Channel connections:
    - Fibre Channel Host Bus Adapters (FC HBA) must be installed and available on cluster nodes (`/sys/class/fc_host/host*`).
    - In the Storage Area Network (SAN), LUN zoning and masking must be configured, providing node initiators with access to the storage target ports.
    - Necessary LUNs must be pre-provisioned on the storage system. DP does not create LUNs.
    - The `multipath-tools` package must be installed on nodes.

Quick start

All commands should be executed on a machine with access to the Kubernetes API and administrator rights.

Enabling the csi-scsi-generic module

Enable the `csi-scsi-generic` module using the command below. This will ensure that the following happens on all cluster nodes:

- The CSI driver is registered.
- Auxiliary pods for the `csi-scsi-generic` components are running.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-scsi-generic
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state.

```
d8 k get module csi-scsi-generic -w
```

Creating an SCSITarget

The SCSITarget resource describes connection to a single SCSI target. When creating a resource in `spec`, specify one of the connection types: `iSCSI` or `fibreChannel`.

iSCSI

The following is a configuration example for connecting via iSCSI. In this example, two SCSITarget resources are created.

You can create multiple resources for either the same or different storage systems. This lets you use multipath to improve failover and performance.

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-1
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx1
    portals:
    - 192.168.1.1

---
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-2
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx2
    portals:
    - 192.168.1.2
EOF
```

To verify that the object has been created, run:

```
d8 k get scsitargets.storage.deckhouse.io <name-of-scsitarget>
```

An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Fibre Channel

To configure a connection based on Fibre Channel, do the following:

1. Before creating the SCSITarget resource, configure zoning and LUN masking on the SAN so that cluster nodes can access the necessary LUNs.
2. Create the SCSITarget resource. In the `spec.fibreChannel.WWNs` field, define a World Wide Port Name (WWPN) of the target storage ports, which expose the necessary LUNs. DP discovers devices by matching the defined WWPNs against paths under `/dev/disk/by-path/`.

   Define the WWPNs in one of the following formats:

   - 16 hexadecimal characters (`2001c89f1acd6117`)
   - With colons (`20:01:c8:9f:1a:cd:61:17`)
   - With the `0x` prefix (`0x2001c89f1acd6117`)

   DP triggers an FC host scan during discovery and volume attach. It does not configure the commutators or log in to targets.

Configuration example with two target storage system ports for multipath:

```shell kubectl apply -f -«EOF apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSITarget metadata: name: hpe-3par-fc-1 spec: deviceTemplate: metadata: labels: my-key: some-label-value fibreChannel: WWNs: - 2001c89f1acd6117

---

apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSITarget metadata: name: hpe-3par-fc-2 spec: deviceTemplate: metadata: labels: my-key: some-label-value fibreChannel: WWNs: - 2001c89f1acd6118 EOF

```
1. Before applying the resource verify that FC HBA are available on the node and that LUN paths are visible after SAN zoning:

   ```shell
   # FC hosts present.
   ls /sys/class/fc_host/

   # Target WWPNs and LUNs available after zoning.
   ls -l /dev/disk/by-path/ | grep -E 'fc-|/fc-'
```

Possible results:

- If the `/sys/class/fc_host/` directory is empty, the node has no FC HBA or the corresponding driver is not loaded.
- If there are no `fc-*` entries in the `/dev/disk/by-path/` directory, check zoning and LUN masking settings on the storage system.
- If the FC paths are displayed, the node is ready to detect LUN provided via Fibre Channel.

1. After the SCSITarget is created, the controller discovers available LUNs and creates SCSIDevice objects. Use the same SCSIStorageClass resource and `scsiDeviceSelector` workflow as for iSCSI.

2. To verify that the object has been created, run the following command:

   ```
   d8 k get scsitargets.storage.deckhouse.io <name-of-scsitarget>
   ```

   An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Creating a StorageClass

To create a StorageClass, use the SCSIStorageClass resource. A command example to create such a resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSIStorageClass
metadata:
  name: scsi-all
spec:
  scsiDeviceSelector:
    matchLabels:
      my-key: some-label-value
  reclaimPolicy: Delete
EOF
```

The `scsiDeviceSelector` field is used to select the SCSITarget for PV creation based on labels. In the example above, all SCSITargets with the label `my-key: some-label-value` are selected. This label will be applied to all devices detected within the specified SCSITarget.

To verify that the object has been created, run:

```
d8 k get scsistorageclasses.storage.deckhouse.io <name-of-scsistorageclass>
```

An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Csi-scsi-generic module health check

Check the status of pods in the `d8-csi-scsi-generic` namespace using the command below. All pods should be in the `Running` or `Completed` state and deployed on all nodes.

```
d8 k -n d8-csi-scsi-generic get pod -owide -w
```

Additional resources

- Module `csi-scsi-generic` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

TATLIN.UNIFIED (Yadro) unified storage

Deckhouse supports integration with the TATLIN.UNIFIED (Yadro) storage system, enabling volume management in Kubernetes. This allows the use of centralized storage for containerized workloads, ensuring high performance and fault tolerance.

This page provides instructions on connecting TATLIN.UNIFIED (Yadro) to Deckhouse, configuring the connection, creating a StorageClass, and verifying system functionality.

Enabling the module

To manage volumes based on the TATLIN.UNIFIED (Yadro) storage system in Deckhouse, the `csi-yadro-tatlin-unified` module is used. It allows the creation of StorageClass resources through custom resources like YadroTatlinUnifiedStorageClass. To enable the module, run the following command:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-yadro-tatlin-unified
spec:
  enabled: true
  version: 1
EOF
```

Wait until `csi-yadro-tatlin-unified` is in the `Ready` status. To check the status, run the following command:

```
d8 k get module csi-yadro-tatlin-unified -w
```

In the output, you should see information about the module:

```
NAME                        STAGE   SOURCE    PHASE       ENABLED    READY
si-yadro-tatlin-unified             Embedded  Available   True       True
```

Connecting to the TATLIN.UNIFIED storage system

To connect to the `TATLIN.UNIFIED` storage system and enable configuring of StorageClass objects, apply the following YadroTatlinUnifiedStorageConnection resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageConnection
metadata:
  name: yad1
spec:
  controlPlane:
    address: "172.19.28.184"
    username: "admin"
    password: "cGFzc3dvcmQ=" # Must be encoded in Base64.
    ca: "base64encoded"
    skipCertificateValidation: true
  dataPlane:
    protocol: "iscsi"
    iscsi:
      volumeExportPort: "p50,p51,p60,p61"
EOF
```

Creating a StorageClass

To create a StorageClass, use the YadroTatlinUnifiedStorageClass resource. Creating a StorageClass resource manually without using YadroTatlinUnifiedStorageClass can lead to errors.

Example command to create a StorageClass based on the `TATLIN.UNIFIED` storage system:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageClass
metadata:
  name: yad1
spec:
  fsType: "xfs"
  pool: "pool-hdd"
  storageConnectionName: "yad1"
  reclaimPolicy: Delete
EOF
```

Ensuring the module works

To make sure the `csi-yadro-tatlin-unified` module is working properly, check the pod status in the `d8-csi-yadro-tatlin-unified` namespace. All pods must have the `Running` or `Completed` status. The `csi-yadro-tatlin-unified` pods must be running on all nodes.

To check that the module works, run the following command:

```
d8 k -n d8-csi-yadro-tatlin-unified get pod -owide -w
```

Additional resources

- Module `csi-yadro-tatlin-unified` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtualization

Deckhouse Platform (DP) runs virtual machines (VMs) in the same cluster as container workloads and

manages them declaratively, through Kubernetes resources. Machines run on the same nodes, networks, and storage as the other workloads, so an administrator configures them with the usual DP tools.

DP lets you do the following:

- Prepare cluster nodes for running virtual machines.
- Store cluster-wide images and provide projects with storage for disks.
- Define the machine parameters that project owners can use.
- Pass node devices through to machines.

Installation and configuration

Virtualization has its own node requirements: hardware virtualization support, the same kernel version everywhere, storage for disks and images, and dedicated subnets for machines. They are listed in Requirements for running virtual machines, while enabling virtualization and placing its components across nodes is covered in Installing virtualization.

The parameters set in the ModuleConfig resource are covered in the module settings reference, and the specifics of updating the components are covered in Updating virtualization.

As guest systems, DP supports operating systems running on the `x86` and `x86-64` architectures. For paravirtualization mode, install the `VirtIO` drivers in the guest system. Linux machines are easier to prepare from images with cloud-init support, and Windows machines with an autounattend answer file.

Storing images and disks

Machine images and disks take up space in the cluster, and an administrator configures how they are stored.

DP keeps uploaded images in its own DVCR storage. Its size, storage class, and the schedule for cleaning up stale data are covered in Virtual machine image storage. The storage classes that projects create disks and images from are listed in Storage for virtual machine disks and images.

Cluster-wide images are prepared by an administrator: project owners create disks from them without uploading the same data again. How to create such images and in what form to store them is covered in Cluster images of virtual machines.

Virtual machine classes

A class defines which processor the guest system sees, which nodes a machine runs on, and which combinations of cores and memory are available to it. A project owner picks a class out of the ones prepared by an administrator, so classes are how DP limits machine parameters in the cluster.

The VirtualMachineClass resource and the purpose of the default class are covered in Virtual machine classes. Each group of settings then has a section of its own: Virtual processor of virtual machines, Placement of virtual machines across nodes, and Virtual machine sizing policy. CPU oversubscription for virtual machines, the share of a physical core that a machine gets guaranteed, is covered separately.

Device passthrough

A machine can be given a node device: a USB device, a GPU, or a PCI device. Each type has its own node requirements and its own preparation order, so each has a section of its own.

Operating a cluster with virtual machines

Running machines also affect other areas of DP configuration:

- Virtual machine networking: Subnets that machines get addresses from.
- Maintenance of nodes running virtual machines: Taking a node out for maintenance without losing machines.
- Virtual machine fault tolerance and balancing: Rebalancing and ColdStandby.
- Virtualization event audit: The log of actions on virtualization resources.

For how virtualization works internally, refer to the Virtualization subsystem section; for the full list of resources and their parameters, refer to the reference. The Virtual machines section is addressed to a project owner.

Additional resources

- Module `virtualization` parameters
- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual machine image storage

Deckhouse Platform (DP) stores virtual machine images in an internal container image storage (DVCR) that resides on a persistent volume of the cluster. Images travel from there to virtual machine disks, so the size of the volume determines how many images fit into the cluster.

Size and storage class

Set the volume size and the storage class in the `.spec.settings.dvcr.storage` block. To expand the storage, increase the volume size.

> ⚠ After the volume is created, you can’t reduce its size or change its storage class.

Cleaning up image storage

When images and disks are deleted from the cluster, their data remains in DVCR for some time. To keep the storage from filling up with stale data, DP runs garbage collection on a schedule. By default, it runs daily at 02:00. To set your own schedule, use the `.spec.settings.dvcr.gc.schedule` parameter in the `virtualization` ModuleConfig.

While garbage collection is running, the storage works in read-only mode, so creating images and disks is postponed until it completes.

- Using the CLI
- Using the web interface

Example configuration with a custom schedule:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: virtualization
spec:
  # ...
  settings:
    dvcr:
      gc:
        schedule: "0 20 * * *"
  # ...
```

To see how much space is occupied and which data will be removed during the next collection, run:

```
d8 k -n d8-virtualization exec deploy/dvcr -- dvcr-cleaner gc check
```

Example output:

```
Found 2 cvi, 5 vi, 1 vd manifests in registry
Found 1 cvi, 5 vi, 11 vd resources in cluster
  Total     Used    Avail     Use%
36.3GiB  13.1GiB  22.4GiB      39%
Images eligible for cleanup:
KIND                   NAMESPACE            NAME
ClusterVirtualImage                         debian-12
VirtualDisk            default              debian-10-root
VirtualImage           default              ubuntu-2404
```

1. Go to the **System** tab, then to **Deckhouse → Modules**.
2. Select the `virtualization` module from the list.
3. In the window that opens, on the **Configuration** tab, enable the **Advanced settings** toggle.
4. In the **Disk and ISO image storage** block, set the schedule in the **Cleanup schedule in Cron format** field.
5. Click **Save**.

Additional resources

- Global custom resources
- Module `virtualization` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage for virtual machine disks and images

The project owner chooses the storage class for an image or a disk. You can limit that choice and set a default class. The `.spec.settings.virtualImages` block covers images, and the `.spec.settings.virtualDisks` block covers disks.

Example:

```
spec:
  settings:
    virtualImages:
      allowedStorageClassSelector:
        matchNames:
          - sc-1
          - sc-2
      defaultStorageClassName: sc-1
    virtualDisks:
      allowedStorageClassSelector:
        matchNames:
          - sc-3
      defaultStorageClassName: sc-3
```

Both blocks work the same way and both are optional. The `allowedStorageClassSelector.matchNames` parameter lists the classes allowed in the VirtualImage and VirtualDisk specification, and `defaultStorageClassName` sets the class for resources where the `.spec.persistentVolumeClaim.storageClassName` parameter isn't set.

Additional resources

- Module `virtualization` parameters
- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster images of virtual machines

An image holds the contents of a disk that project owners use to create virtual machine disks. A cluster image, ClusterVirtualImage, is available in every namespace and project of the cluster, so an image uploaded once serves all projects at once.

An image appears in the cluster in three steps:

1. The administrator creates a ClusterVirtualImage resource and specifies a data source in it.
2. Deckhouse Platform (DP) downloads the image from that source to the internal storage (DVCR).
3. The downloaded image becomes available for creating disks.

The image source can be an HTTP server hosting the image file, a container image registry, or a file on your computer that you upload from the command line. You can also create an image from another image, from a virtual machine disk, or from a disk snapshot.

The `PHASE` column in the `d8 k get cvi` output shows the progress of image creation; for its values, see the `.status.phase` field. To follow the creation in real time, add the `-w` flag. If an image stays not ready for a long time, check the `.status.conditions` block and the `d8 k describe cvi` output for the reason.

Until an image reaches the `Ready` phase, you can change its `.spec` block, and the download restarts after each change. For a ready image, the `.spec` block can no longer be changed. For all image parameters, see ClusterVirtualImage.

Image types and formats

There are two types of images:

- **ISO image**: An installation image used for the initial installation of an operating system (OS). OS vendors publish such images and use them to install the OS on physical and virtual servers.
- **Disk image with a preinstalled system**: Contains an OS that is already installed and configured, and is ready to work as soon as the virtual machine (VM) is created. Distribution vendors publish such images, or you can build them yourself.

Distribution vendors publish ready-made images with a preinstalled system. The following table lists their download pages and the users configured in those images by default:

Distribution	Default user
AlmaLinux	`almalinux`
AlpineLinux	`alpine`
AltLinux	`altlinux`
AstraLinux	`astra`
CentOS	`cloud-user`
Debian	`debian`
Rocky	`rocky`
Ubuntu	`ubuntu`

DP accepts image files in the following formats:

- `qcow2`
- `raw`
- `vmdk`
- `vdi`
- `vhd`
- `vhdx`

You can provide an image compressed with `gz`, `xz`, or `zst`. DP unpacks it during the upload.

DP detects the image type and size on its own and records them in the resource status. There are two sizes, and both appear in the `d8 k get cvi -o wide` output:

- `STOREDSIZE`: The space the image occupies in the storage. For an image uploaded in a compressed form, it's smaller than the unpacked size. Use this column to estimate how much space the images take in DVCR.
- `UNPACKEDSIZE`: The size of the image after unpacking. It defines the minimum size of a disk that can be created from this image.

> **i** When creating a disk from an image, specify a size no smaller than the `UNPACKEDSIZE` value. If you don't specify a size, the disk is created with exactly the unpacked size of the image.

Creating a cluster image from an HTTP server

The simplest way to create an image is to provide a link to a file hosted on an HTTP server.

- Using the CLI
- Using the web interface

1. Create a ClusterVirtualImage resource:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: ClusterVirtualImage
metadata:
  name: ubuntu-24-04
spec:
  # Source for the image.
  dataSource:
    type: HTTP
    http:
      url: https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-
amd64.img
EOF
```

2. Verify that the image is created:

```
d8 k get clustervirtualimage ubuntu-24-04

# Short form of the command.
d8 k get cvi ubuntu-24-04
```

   Example output:

```
NAME           PHASE   CDROM   PROGRESS   AGE
ubuntu-24-04   Ready   false   100%       23h
```

To make DP verify the downloaded file against a checksum, add the `checksum` block to the source. If the file doesn't match any of the specified checksums, the image moves to the `Failed` phase.

1. Go to the **System** tab, then to **Virtualization → Cluster images**.
2. Click **Create**, then select **By link** in the **Source** block.
3. In the **Image name** field, enter the image name.
4. In the **URL** field, specify the link to the image.
5. Click **Create**.
6. Wait until the image reaches the **Ready** state.

Creating a cluster image from a container image registry

DP can pull an image from an external container image registry, but the disk file must be located in the container image under the `/disk` path. The following steps show how to prepare such a container image and create a cluster image from it.

- Using the CLI
- Using the web interface

1. Download the image file to your local machine:

   ```
   curl -L https://cloud-images.ubuntu.com/minimal/releases/noble/release/ubuntu-24.04-minimal-cloudimg-amd64.img -o ubuntu2404.img
   ```

2. Create a `Dockerfile` with the following contents:

   ```
   FROM scratch
   COPY ubuntu2404.img /disk/ubuntu2404.img
   ```

3. Build the container image. The example uses the docker.com registry, which requires an account and a configured environment:

   ```
   docker build -t docker.io/<USERNAME>/ubuntu2404:latest
   ```

   Where `<USERNAME>` is the username you specified when registering in the registry.

4. Push the built container image to the registry:

   ```
   docker push docker.io/<USERNAME>/ubuntu2404:latest
   ```

5. Create a ClusterVirtualImage resource that points to the container image:

   ```
   d8 k apply -f - <<EOF
   apiVersion: virtualization.deckhouse.io/v1alpha2
   kind: ClusterVirtualImage
   metadata:
     name: ubuntu-2404
   spec:
     dataSource:
       type: ContainerImage
       containerImage:
         image: docker.io/<USERNAME>/ubuntu2404:latest
   EOF
   ```

DP works only with registries that have TLS enabled. If the registry uses its own certificate authority, provide the certificate chain in the `caBundle` parameter, and take the credentials for a private registry from the secret specified in the `imagePullSecret` parameter.

1. Go to the **System** tab, then to **Virtualization** → **Cluster images**.
2. Click **Create**, then select **From registry** in the **Source** block.
3. In the **Image name** field, enter the image name.
4. In the **Image in container registry** field, specify the link to the image.
5. Click **Create**.
6. Wait until the image reaches the **Ready** state.

Uploading a cluster image from the command line

If the image file is on your computer, upload it directly. DP creates a temporary upload endpoint for this and waits for the data.

- Using the CLI
- Using the web interface

1. Create a ClusterVirtualImage resource with the `Upload` source:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: ClusterVirtualImage
metadata:
  name: some-image
spec:
  # Image source settings.
  dataSource:
    type: Upload
EOF
```

The resource moves to the `WaitForUserUpload` phase and is ready to accept the file. Start the upload within 10 minutes, otherwise the resource moves to the `Failed` phase and you have to create it again.

2. Get the addresses that accept the file:

   ```
   d8 k get cvi some-image -o jsonpath="{.status.imageUploadURLs}" | jq
   ```

   Example output:

   ```
   {
     "external":"https://virtualization.example.com/upload/<SECRET_URL>",
     "inCluster":"http://10.222.165.239/upload"
   }
   ```

   Use the `inCluster` address if you upload the file from one of the cluster nodes, and `external` in all other cases.

3. Upload the file to the selected address. The example first downloads the Cirros image and then sends it to the cluster:

   ```
   curl -L http://download.cirros-cloud.net/0.5.1/cirros-0.5.1-x86_64-disk.img -o cirros.img
   curl https://virtualization.example.com/upload/<SECRET_URL> --progress-bar -T cirros.img | cat
   ```

   Where `<SECRET_URL>` is the last part of the address from the previous step.

4. Verify that the image has reached the `Ready` phase:

   ```
   d8 k get cvi some-image
   ```

   Example output:

   ```
   NAME         PHASE   CDROM   PROGRESS   AGE
   some-image   Ready   false   100%       1m
   ```

You can also verify the uploaded file against a checksum. To do this, specify the `checksum` block in the data source.

1. Go to the **System** tab, then to **Virtualization → Cluster images**.
2. Click **Create**, then select **Upload** in the **Source** block.
3. In the **Image name** field, enter the image name.
4. In the **Upload file** block, drag the file to the highlighted area or click **select on your computer**.
5. Select the file in the file manager that opens.
6. Click **Create**.
7. Wait until the image reaches the **Ready** state.

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual machine classes

A virtual machine (VM) class defines what the project owner doesn't configure: the virtual CPU model, the allowed combinations of cores and memory, and the nodes where a VM can run. These rules are described by the VirtualMachineClass resource, which you use to control how project workloads are distributed across cluster nodes.

On the initial installation, the module creates the `generic` class with the Nehalem CPU model. This model is old but supported by any modern CPU, so VMs of this class start on any cluster node and migrate between nodes without restrictions.

> The `generic` class matches a CPU with the smallest instruction set, so it isn't suitable for production workloads.
>
> Once all nodes are added to the cluster and configured, create at least one class with the `Discovery` CPU type. Deckhouse Platform (DP) selects an instruction set available on all nodes at once, so virtual machines can make fuller use of the CPUs while still being able to migrate between nodes. The instruction set is fixed when the resource is created and doesn't change as nodes are added or removed.
>
> For an example of such a class, see vCPU Discovery configuration example.

Classes exist at the cluster level. To list them, run the following command:

```
d8 k get virtualmachineclass
```

Example output:

```
NAME      PHASE   ISDEFAULT   AGE
generic   Ready               6d1h
```

In any class, you can change everything except the `.spec.cpu` block, because the CPU model is fixed when the resource is created. You can both modify and delete the `generic` class, but it won't be created again, because the module adds it only on the initial installation.

The project owner specifies the class in the `.spec.virtualMachineClassName` parameter of a virtual machine:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: linux-vm
spec:
  virtualMachineClassName: generic # Name of the VirtualMachineClass resource.
  # ...
```

Default VirtualMachineClass

You can designate one of the classes as the default. DP inserts its name into the `.spec.virtualMachineClassName` parameter if the project owner doesn't specify a class.

The default class is marked with the `virtualmachineclass.virtualization.deckhouse.io/is-default-class` annotation set to `true`. A cluster can have only one such class, so to designate a new one, first remove the annotation from the current one.

Don't add the annotation to the `generic` class, because a module update can remove it. Create your own class and designate it as the default instead.

1. Check which classes exist in the cluster:

   ```
   d8 k get vmclass
   ```

   Example output with no default class:

   ```
   NAME                      PHASE   ISDEFAULT   AGE
   generic                   Ready               1d
   host-passthrough-custom   Ready               1d
   ```

2. Designate the default class:

   ```
   d8 k annotate vmclass host-passthrough-custom virtualmachineclass.virtualization.deckhouse.io/is-default-class=true
   ```

3. Verify that the annotation is set:

   ```
   d8 k get vmclass
   ```

   Example output:

   ```
   NAME                      PHASE   ISDEFAULT   AGE
   generic                   Ready               1d
   host-passthrough-custom   Ready   true        1d
   ```

From now on, virtual machines created without a class get the `host-passthrough-custom` class.

VirtualMachineClass settings

A class consists of three blocks, each responsible for its own group of settings:

- Using the CLI
- Using the web interface

Describe the class in a VirtualMachineClass resource:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: <VMCLASS_NAME>
  # The annotation designates the class as the default one. It's optional.
  # annotations:
  #   virtualmachineclass.virtualization.deckhouse.io/is-default-class: "true"
spec:
  # Virtual CPU parameters. The block is required and can't be changed after the
resource is created.
  cpu: ...

  # Rules for placing virtual machines on nodes. The block is optional.
  # Changes apply to all VMs of this class.
  nodeSelector: ...

  # Resource allocation policy for virtual machines. The block is optional.
  # Changes apply to all VMs of this class.
  sizingPolicies: ...
```

Where `<VMCLASS_NAME>` is the name of the class you're creating.

1. Go to the **System** tab, then to **Virtualization** → **VM classes**.
2. Click **Create**.
3. In the form that opens, enter the VM class name in the **Name** field.

The blocks are described separately in the following sections.

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual processor of virtual machines

A virtual machine (VM) class defines which processor and which instruction set the guest system sees.

Virtual processor

The `.spec.cpu` block defines the CPU that the guest OS sees. It also determines which nodes a VM can migrate between.

> ⚠ The `.spec.cpu` block can't be changed after the resource is created. To use a different CPU, create a new class.

The following examples cover each CPU type.

- A set of CPU instructions required for a VM. Set with the `Features` type:

  ```
  spec:
    cpu:
      features:
        - vmx
      type: Features
  ```

  To configure the vCPU in the web interface, in the VM class creation form:

  1. In the **CPU settings** block, select `Features` in the **Type** field.
  2. In the **Required set of supported instructions** field, select the instructions you need.
  3. Click **Create**.

- A universal CPU for a given set of nodes. Set with the `Discovery` type:

  ```
  spec:
    cpu:
      discovery:
        nodeSelector:
          matchExpressions:
            - key: node-role.kubernetes.io/control-plane
              operator: DoesNotExist
      type: Discovery
  ```

  To do the same in the web interface, in the VM class creation form:

  1. In the **CPU settings** block, select `Discovery` in the **Type** field.
  2. Click **Add** in the **Conditions for creating a universal CPU → Labels and expressions** block.
  3. Set **Key**, **Operator**, and **Value**. They correspond to the `.spec.cpu.discovery.nodeSelector` parameter.
  4. Press **Enter** to confirm the key parameters.
  5. Click **Create**.

- A CPU close to the node CPU. Set with the `Host` type. The guest OS gets almost the full instruction set of the node, so performance is higher than with a fixed model. A VM of this class migrates only between nodes with similar CPUs. For example, migration between nodes with Intel and AMD CPUs isn't possible, and neither is migration between CPU generations with different instruction sets.

```
spec:
  cpu:
    type: Host
```

To do the same in the web interface, in the VM class creation form:

1. In the **CPU settings** block, select `Host` in the **Type** field.
2. Click **Create**.

- The node CPU without changes. Set with the `HostPassthrough` type. A VM of this class migrates only to a node whose CPU exactly matches the CPU of the source node.

```
spec:
  cpu:
    type: HostPassthrough
```

To do the same in the web interface, in the VM class creation form:

1. In the **CPU settings** block, select `HostPassthrough` in the **Type** field.
2. Click **Create**.

- A specific CPU model with a known instruction set. Set with the `Model` type. First, check which models the target node supports:

```
d8 k get nodes <NODE_NAME> -o json | jq '.metadata.labels | to_entries[] | select(.key
| test("cpu-model.node.virtualization.deckhouse.io")) | .key | split("/")[1]' -r
```

Where `<NODE_NAME>` is the name of a cluster node.

Example output:

```
Broadwell-noTSX
Broadwell-noTSX-IBRS
Haswell-noTSX
Haswell-noTSX-IBRS
IvyBridge
IvyBridge-IBRS
Nehalem
Nehalem-IBRS
Penryn
SandyBridge
SandyBridge-IBRS
Skylake-Client-noTSX-IBRS
Westmere
Westmere-IBRS
```

Then specify the selected model in the class specification:

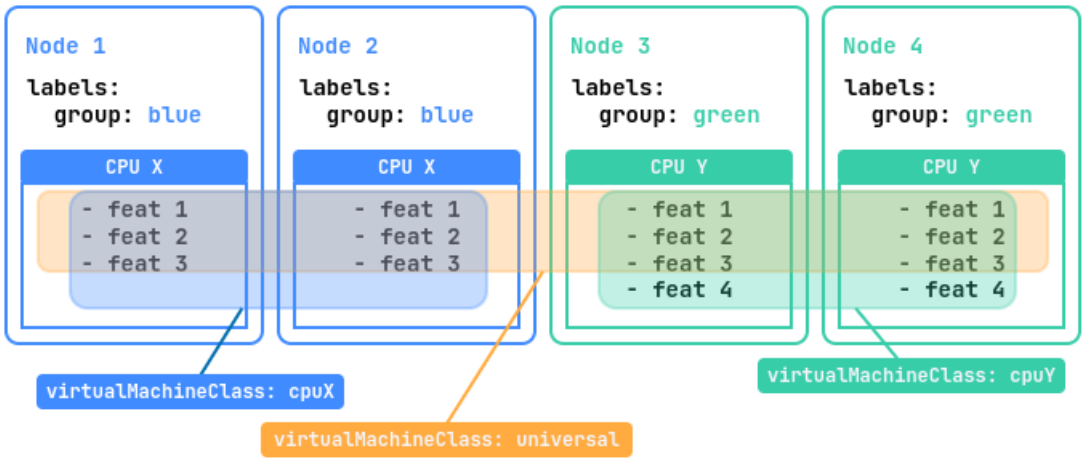
```
spec:
  cpu:
    model: IvyBridge
    type: Model
```

To do the same in the web interface, in the VM class creation form:

1. In the **CPU settings** block, select `Model` in the **Type** field.
2. In the **Model** field, select the CPU model.
3. Click **Create**.

vCPU Discovery configuration example

The following example shows how to choose the processor types in a cluster with heterogeneous nodes.



The example below uses a cluster of four nodes. Two nodes labeled `group=blue` are equipped with the “CPU X” processor with three instruction sets, and the other two labeled `group=green` have the newer “CPU Y” processor with four sets.

> **i** A CPU instruction set is every command the processor can execute, from addition to memory operations. The set determines which programs run and how fast, and it differs between CPU generations.

Three classes suit such a cluster:

- `universal`: VMs start on any node and migrate between all four. Deckhouse Platform (DP) takes the instruction set common to both processors, so compatibility is maximal, while some capabilities of “CPU Y” stay unused.
- `cpuX`: VMs start only on nodes with “CPU X” and migrate between them, using all instructions of that processor.
- `cpuY`: The same for nodes with “CPU Y”.

The classes for such a cluster look as follows:

```
---
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: universal
spec:
  cpu:
    # An empty discovery means that all cluster nodes are taken into account.
    discovery: {}
    type: Discovery
---
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: cpuX
spec:
  cpu:
    discovery:
      nodeSelector:
        matchExpressions:
          - key: group
            operator: In
            values: ["blue"]
    type: Discovery
---
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: cpuY
spec:
  cpu:
    discovery:
      nodeSelector:
        matchExpressions:
          - key: group
            operator: In
            values: ["green"]
    type: Discovery
```

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Placement of virtual machines across nodes

The optional `.spec.nodeSelector` block limits the set of nodes where virtual machines (VMs) of this class run. Nodes are selected by labels:

```
spec:
  nodeSelector:
    matchExpressions:
      - key: node.deckhouse.io/group
        operator: In
        values:
          - green
```

⚠ A change to the `.spec.nodeSelector` block affects all virtual machines of the class at once. Those running on nodes that no longer match the new conditions have to be moved:

- In commercial editions, Deckhouse Platform (DP) migrates such VMs to suitable nodes.
- In DP Open, the VMs are restarted, and the restart time depends on the `.spec.disruptions.restartApprovalMode` parameter of the virtual machine, which defaults to `Manual` and requires the project owner's approval.

To do the same in the web interface, in the VM class creation form:

1. Click **Add** in the **Conditions for scheduling VMs on nodes → Labels and expressions** block.
2. Set **Key**, **Operator**, and **Value**. They correspond to the `.spec.nodeSelector` parameter.
3. Press **Enter** to confirm the key parameters.
4. Click **Create**.

Additional resources

- Module `virtualization` custom resources

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtual machine sizing policy

The `.spec.sizingPolicies` block defines which combinations of cores, core fraction, and memory are allowed for virtual machines (VMs) of this class.

⚠ Changes to the `.spec.sizingPolicies` block affect existing virtual machines. For virtual machines that no longer meet the new requirements, the `SizingPolicyMatched` condition in the `.status.conditions` block gets the `False` status.

When defining policies, take the CPU topology of virtual machines into account.

A policy consists of a list of rules, each applying to its own range of cores. The range is set in the required `cores` block, and ranges of different rules must not overlap, otherwise Deckhouse Platform (DP) rejects the class.

A valid structure, where the ranges follow one another without overlapping:

```
- cores:
    min: 1
    max: 4
  # ...
- cores:
    min: 5 # The next range starts one above the previous max.
    max: 8
```

An invalid structure, where the value `4` falls into two ranges at once:

```
- cores:
   min: 1
   max: 4
  # ...
- cores:
   min: 4
   max: 8
```

DP doesn’t forbid gaps between ranges, but a virtual machine whose number of cores falls outside every range is left without a policy. For this reason, start each range with the value that follows the `max` of the previous one.

Within a range, you set the requirements for memory and for the core fraction:

- `memory`: The minimum and maximum amount of memory. Set either for the whole range or per core, in the nested `memory.perCore` block.
- `coreFractions`: The list of allowed core fractions, for example `[25, 50, 100]` for 25%, 50%, and 100%. If the project owner sets the `coreFraction` parameter of a virtual machine explicitly, the value must come from this list.
- `defaultCoreFraction`: The core fraction that a virtual machine gets if `coreFraction` isn’t set in it. The value must be in the `coreFractions` list. If the parameter isn’t specified, 100% applies.

A rule with neither `memory` nor `coreFractions` doesn’t limit anything, so set at least one of them.

In commercial DP editions, you can set the `defaultCoreFraction` parameter to `Auto`. In that case, the core fraction for VMs without an explicit `coreFraction` is chosen by vertical autoscaling. `Auto` is a mode, not a share of a core, so it must not appear in the `coreFractions` list.

```
spec:
  sizingPolicies:
    - cores:
       min: 1
       max: 8
      coreFractions: [10, 25, 50, 100]
      defaultCoreFraction: Auto
```

The `Auto` value is accepted only when both capabilities are available:

- Vertical autoscaling of virtual machines, which is enabled automatically in commercial DP editions when the `vertical-pod-autoscaler` module is enabled.
- Changing the number of cores and the amount of memory without a restart, which is enabled by the `HotplugCPUAndMemoryWithInPlaceResize` feature in the `.spec.settings.featureGates` parameter.

If at least one of them is unavailable, DP rejects the creation of such a class.

A new default applies only to the virtual machines created after the change. An existing machine already carries the previous value in its specification, and the value stays there until the project owner sets another one.

The following examples show how the amount of memory depends on the number of cores:

- The `memory` parameter sets the same limits for the whole range of cores:

  ```
  - cores:
     min: 1
     max: 4
    memory:
     min: 2Gi
     max: 8Gi
  ```

  A virtual machine with any number of cores from 1 to 4 gets from 2 to 8 GiB of memory, and the number of cores doesn’t affect these limits.

- The `memory.perCore` parameter sets the limits per core, and the resulting limits are the product of that value and the number of cores:

```
- cores:
    min: 1
    max: 4
  memory:
    perCore:
      min: 1Gi
      max: 2Gi
```

With such a policy, the allowed amount of memory grows with the number of cores:

- 1 core: From 1 to 2 GiB.
- 2 cores: From 2 to 4 GiB.
- 3 cores: From 3 to 6 GiB.
- 4 cores: From 4 to 8 GiB.

- The `memory.step` parameter limits the allowed memory values to a grid, so that the project owner can’t choose arbitrary amounts.

  Together with `memory.min` and `memory.max`, the step is counted from the minimum:

  ```
  - cores:
      min: 1
      max: 4
    memory:
      min: 2Gi
      max: 8Gi
      step: 1Gi
  ```

  Only the values 2, 3, 4, 5, 6, 7, and 8 GiB are allowed; 2.5 or 7.5 GiB can’t be set.

  Together with `memory.perCore`, the step is counted from the memory per core, and the resulting value is then multiplied by the number of cores:

  ```
  - cores:
      min: 1
      max: 4
    memory:
      perCore:
        min: 1Gi
        max: 2Gi
      step: 512Mi
  ```

  Per core, 1, 1.5, and 2 GiB are allowed, so the resulting amount depends on the number of cores:

  - 1 core: 1, 1.5, or 2 GiB.
  - 2 cores: 2, 3, or 4 GiB.
  - 3 cores: 3, 4.5, or 6 GiB.
  - 4 cores: 4, 6, or 8 GiB.

An example of a policy that covers ranges from 1 to 248 cores:

```
spec:
  sizingPolicies:
    # For 1-4 cores, from 1 to 8 GiB of memory is available with a 512 MiB step,
    # that is, 1 GiB, 1.5 GiB, 2 GiB, 2.5 GiB, and so on.
    # All core fractions are available.
    - cores:
        min: 1
        max: 4
      memory:
        min: 1Gi
        max: 8Gi
        step: 512Mi
      coreFractions: [5, 10, 20, 50, 100]
      defaultCoreFraction: 50 # Default core fraction for the 1-4 core range.
    # For 5-8 cores, from 5 to 16 GiB of memory is available with a 1 GiB step,
    # that is, 5 GiB, 6 GiB, 7 GiB, and so on.
    # Core fractions are limited to three values.
    - cores:
        min: 5
        max: 8
      memory:
        min: 5Gi
        max: 16Gi
        step: 1Gi
      coreFractions: [20, 50, 100]
      defaultCoreFraction: 100 # Default core fraction for the 5-8 core range.
    # For 9-16 cores, from 9 to 32 GiB of memory is available with a 1 GiB step.
    # Core fractions are limited to two values.
    - cores:
        min: 9
        max: 16
      memory:
        min: 9Gi
        max: 32Gi
        step: 1Gi
      coreFractions: [50, 100]
    # For 17-248 cores, from 1 to 2 GiB of memory is available per core.
    # The core fraction is 100% only.
    - cores:
        min: 17
        max: 248
      memory:
        perCore:
          min: 1Gi
          max: 2Gi
      coreFractions: [100]
```

To configure sizing policies in the web interface, in the VM class creation form:

1. Click **Add** in the **Resource allocation rules for virtual machines** block.
2. In the **CPU** block, enter `1` in the **Min** field and `4` in the **Max** field.
3. In the **CPU** block, in the **Allow setting core fractions** field, select the values `5%`, `10%`, `20%`, `50%`, `100%` in order.
4. In the **Memory** block, set the toggle to **Amount per 1 core**.
5. In the **Memory** block, enter `1` in the **Min** field and `8` in the **Max** field.
6. In the **Memory** block, enter `1` in the **Discretization step** field.
7. Add other ranges with the **Add** button, if required.
8. Click **Create**.

Additional resources

- Module `vertical-pod-autoscaler` documentation
- Module `virtualization` custom resources

- Module `virtualization` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

CPU oversubscription for virtual machines

Oversubscription lets you give the virtual machines (VMs) on a node more virtual cores than the node physically has. This makes sense because VMs rarely load the CPU at the same time and at full capacity.

The degree of oversubscription is controlled by the `coreFraction` parameter of a virtual machine, and you define its allowed values in the sizing policy of the class. The parameter defines the share of a core's capacity guaranteed to a VM. For example, with `coreFraction: 20%`, a VM always gets a fifth of a core, and it can take a whole core when the node has spare resources.

> If the `coreFractions` list isn't set in the class or contains several values, the project owner chooses the degree of oversubscription by specifying `coreFraction` when creating a VM.

When placing a VM on a node, Deckhouse Platform (DP) sums the guaranteed shares of all VMs on that node using the `Σ(cores × coreFraction / 100)` formula. If the sum exceeds the number of physical cores, the VM doesn't start on that node.

Consider a node with 4 physical cores and 5 VMs, each with 2 cores and `coreFraction: 20%`. The guaranteed load is `5 × 2 × 0.2 = 2` cores, with 10 virtual cores on 4 physical ones, which is an oversubscription of 2.5 to 1. All five VMs fit on the node, because 2 cores is less than the available 4.

Fixed oversubscription

A list with a single value leaves the project owner no choice, and you define the degree of oversubscription for all VMs of the class:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: oversubscribed
spec:
  sizingPolicies:
    - cores:
        min: 1
        max: 8
      memory:
        perCore:
          min: 1Gi
          max: 8Gi
      coreFractions: [20] # The only allowed value.
      defaultCoreFraction: 20
```

All VMs of this class get 20% of a core each, which gives an oversubscription of 5 to 1.

Oversubscription chosen by the project owner

A list with several values leaves the choice to the project owner:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineClass
metadata:
  name: standard
spec:
  sizingPolicies:
    - cores:
        min: 1
        max: 4
      memory:
        perCore:
          min: 1Gi
          max: 8Gi
      coreFractions: [5, 10, 20, 50, 100]
      defaultCoreFraction: 20
```

The project owner selects `coreFraction` from the list, and if they don’t, the VM gets 20%.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

USB devices in virtual machines

USB device passthrough to virtual machines (VMs) is handled by the `virtualization-dra` system component, which needs three kernel modules on the node:

- `usbip_core`
- `usbip_host`
- `vhci_hcd`

DP loads them on the nodes itself. A node where all three modules are available gets the `virtualization.deckhouse.io/usbip=true` label, and the `virtualization-dra` component runs only on such nodes. If the kernel modules stop being available, the label is removed and the component is deleted from the node.

To see which nodes are ready for USB device passthrough, run the following command:

```
d8 k get nodes -l virtualization.deckhouse.io/usbip=true
```

Example output:

```
NAME     STATUS   ROLES    AGE   VERSION
node-1   Ready    worker   10d   v1.34.1
```

To verify that the component is actually running on these nodes, run the following command:

```
d8 k -n d8-virtualization get pods -l app=virtualization-dra -o wide
```

A node missing from the output failed to load the kernel modules, and USB devices on that node aren’t detected. Install the kernel modules yourself from your operating system package, or build them for the kernel in use. DP detects them on its own and assigns the label to the node within a few minutes.

Path of a USB device from a node to a VM

A USB device travels from the node to a virtual machine in four steps:

1. The DRA driver detects USB devices on the nodes and publishes information about them to the Kubernetes API as a ResourceSlice. DP creates NodeUSBDevice resources from this data.
2. The administrator assigns a namespace to the NodeUSBDevice resource by setting the

`.spec.assignedNamespace` parameter. This makes the device available in that namespace.

3. Once the namespace is assigned, DP creates a USBDevice resource in it.
4. The project owner attaches the USBDevice device to a virtual machine by adding it to the `.spec.usbDevices` parameter of the VirtualMachine resource.

Discovered devices (NodeUSBDevice)

The NodeUSBDevice resource describes a physical USB device detected on a node. The resource exists at the cluster level, so you see all detected devices in a single list:

```
d8 k get nodeusbdevice
```

Example output:

```
NAME              NODE     READY   ASSIGNED   ATTACHED   NAMESPACE    AGE
usb-flash-drive   node-1   True    False      False                   10m
logitech-webcam   node-2   True    True       True       my-project   15m
```

The conditions in the `.status.conditions` block reflect the readiness of the device and its state. The `Ready` and `Attached` conditions match the USBDevice conditions, and the `Assigned` condition shows whether a namespace is assigned to the device:

- `Available`: No namespace is assigned.
- `InProgress`: A namespace is assigned and the USBDevice resource is being created.
- `Assigned`: The USBDevice resource is created and the device is available in the namespace.

Assigning a namespace to a USB device

Until a namespace is assigned to a device, the project owner doesn’t see it. To make the device available in a project, follow these steps.

1. Connect the USB device to a node that is ready for passthrough and wait for a NodeUSBDevice resource to appear.
2. Assign the namespace with the `.spec.assignedNamespace` parameter:

   ```
   d8 k apply -f - <<EOF
   apiVersion: virtualization.deckhouse.io/v1alpha2
   kind: NodeUSBDevice
   metadata:
     name: logitech-webcam
   spec:
     assignedNamespace: my-project
   EOF
   ```

3. Verify that a USBDevice resource appears in the namespace:

   ```
   d8 k get usbdevice -n my-project
   ```

After that, the project owner attaches the device to a virtual machine.

Viewing USB device details

Full details about a device and its current state are available in the resource status.

> **i** When a device is physically disconnected from the node, the `Attached` condition gets the `False` value, and the `Ready` condition gets the `NotFound` reason. The same is reflected in the status of the USBDevice resource in the project namespace.

- Using the CLI
- Using the web interface

The device identifiers, its location, and the current conditions are stored in the resource status:

```
d8 k get nodeusbdevice <DEVICE_NAME> -o yaml
```

Where `<DEVICE_NAME>` is the name of the NodeUSBDevice resource.

To get only the device attributes, query the fields you need directly:

```
d8 k get nodeusbdevice <DEVICE_NAME> \
  -o jsonpath='{.status.attributes.manufacturer}{" "}{.status.attributes.product}{"
("}{.status.attributes.vendorID}{":"}{.status.attributes.productID}{")\n"}'
```

Example output:

```
Logitech Webcam C920 (046d:082d)
```

1. Go to the **System** tab, then to **Virtualization → Node USB devices**.
2. Review the list, which shows the device status, manufacturer, product, serial number, node, bus, device number, and assigned namespace.

The project owner sees the devices assigned to their project in the **Virtualization → USB devices** section of that project.

Requirements and limitations

When planning USB device passthrough, consider the following requirements and limitations:

- A node where USB devices must be detected has to carry the `virtualization.deckhouse.io/usbip=true` label and run containerd version 2, otherwise the `virtualization-dra` component doesn't start there.
- A device is passed to a virtual machine over the network using USBIP, so the VM can run on a node other than the one the device is physically connected to.
- Only a device that reports the USB 2.0 speed (480 Mbps) or a USB 3.x speed (5 Gbps and higher) can be passed through. DP doesn't let you attach a slower device to a VM, for example a mouse or a keyboard at 1.5 or 12 Mbps.
- A node connects no more than 16 devices, 8 per USB 2.0 hub and 8 per USB 3.0 hub.
- The hub is selected by the device speed and can't be changed manually. A USB 2.0 device doesn't connect to a USB 3.0 hub, and vice versa.
- A device can be attached to a running VM and detached from it without stopping the VM.

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

GPU devices in virtual machines

DP attaches physical GPU devices to virtual machines using DRA (Dynamic Resource Allocation). A project owner requests a device by a reference to a `GPUClass` in the `.spec.gpus` block of their machine, and you prepare the cluster for this.

To make passthrough work, provide the following:

- Kubernetes 1.34 or later with the DRA feature gates required by your cluster configuration.
- The `GPU` feature gate in the module settings.
- The enabled `gpu` module in the DRA mode, which is set by the `dra.enabled` parameter.
- A `GPUClass` resource that selects devices of the model you need. The `gpu` module creates a DeviceClass resource with the same name from it, and the device is allocated to a machine through that class.

To enable the feature gate, add it to the module settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: virtualization
spec:
  settings:
    featureGates:
      - GPU
```

After that, tell the project owners the names of the available `GPUClass` resources. A single machine takes no more than 16 devices, and a change to the `.spec.gpus` block applies only after the machine restarts.

Additional resources

- Module `gpu` documentation
- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

PCI devices in virtual machines

PCI device passthrough lets you attach a physical device of a node to a virtual machine (VM), for example an industrial controller, a hardware security module, a capture card, an FPGA, or an entire network card. In the guest operating system, such a device works under its own driver, so a VM can use the hardware DP doesn’t support directly.

A device reaches a machine in two steps. First you assign it to a project namespace, and then the project owner lists the device in the specification of their machine. The device is granted for exclusive use, so it’s available in one namespace and to one machine only.

DP switches the device drivers itself, and you don’t need to configure them on the node manually. When a machine starts, DP unbinds the device from the regular kernel driver and binds it to the `vfio-pci` driver, and once the machine is stopped, it returns the device to the regular driver. While the machine is running, the node doesn’t use the device.

Node requirements

PCI device passthrough is handled by the `virtualization-dra-pci` system component. It runs only on nodes with containerd version 2 and hardware I/O virtualization enabled. DP checks the nodes itself and assigns the `virtualization.deckhouse.io/vfio=true` label to the suitable ones.

To enable hardware I/O virtualization, turn on `VT-d` on Intel or `AMD-Vi` on AMD in the node BIOS and add the `intel_iommu=on` or `amd_iommu=on` kernel parameter.

To see which nodes are ready for PCI device passthrough, run the following command:

```
d8 k get nodes -l virtualization.deckhouse.io/vfio=true,virtualization.deckhouse.io/containerd-version=v2
```

Example output:

```
NAME     STATUS   ROLES    AGE   VERSION
node-1   Ready    worker   10d   v1.34.1
```

A node missing from the output has not been assigned the label. Check the `/sys/kernel/iommu_groups` directory on it. An empty directory means that hardware I/O virtualization is disabled. Enable it and reboot the node, and the label is assigned automatically within a few minutes.

To verify that the component is actually running on these nodes, run the following command:

```
d8 k -n d8-virtualization get pods -l app=virtualization-dra-pci -o wide
```

Assigning a namespace to a PCI device

DP detects the devices on the suitable nodes and creates a NodePCIDevice resource for each of them. The component scans the PCI bus at startup and then every five minutes, so a newly installed device appears in the list within a few minutes.

To make a device available to a project, follow these steps.

1. Find the device among the detected ones:

   ```
   d8 k get nodepcidevice
   ```

   Example output:

   ```
   NAME                                              NODE     ADDRESS        READY
   ASSIGNED   ATTACHED   NAMESPACE    AGE
   pci-4f2c0b1e8d9a3c5b7e1f0a2d4c6b8e0f1a3c5d7e    node-1   0000:3b:00.0   True
   False      False                  10m
   pci-9a1b3c5d7e9f0a2b4c6d8e0f1a3b5c7d9e1f0a2b    node-2   0000:65:00.0   True
   True       False      my-project   15m
   ```

   The resource name is a hash of the device parameters and the node name, so look up the device you need by the `NODE` and `ADDRESS` columns. To verify the choice, use the `.status.attributes` block, which holds the PCI bus address and the vendor and model identifiers. They let you find the same device in the `lspci -nn` output on the node.

2. Assign a namespace with the `.spec.assignedNamespace` parameter:

   ```
   d8 k apply -f - <<EOF
   apiVersion: virtualization.deckhouse.io/v1alpha2
   kind: NodePCIDevice
   metadata:
     name: pci-4f2c0b1e8d9a3c5b7e1f0a2d4c6b8e0f1a3c5d7e
   spec:
     assignedNamespace: my-project
   EOF
   ```

3. Make sure that a PCIDevice resource has appeared in the namespace:

   ```
   d8 k get pcidevice -n my-project
   ```

After that, the project owner attaches the device to their machine by specifying the PCIDevice resource name in the `.spec.pciDevices` parameter of the VirtualMachine resource.

To make the device unavailable to the project, clear the `.spec.assignedNamespace` parameter or set a different namespace. The PCIDevice resource in the previous namespace is deleted.

While the device is listed in a machine specification, the PCIDevice resource isn't deleted and the machine keeps running. Ask the project owner to remove the device from the specification. If you set a different namespace instead of clearing the parameter, the resource appears in it right away, but the machine of the new project stays in the `Pending` phase while the device is occupied by the previous machine.

Requirements and limitations for PCI devices

When planning PCI device passthrough, consider the following requirements and limitations:

- Passthrough works in a cluster with Kubernetes 1.34 or higher and containerd version 2 on the nodes, and the `DRAResourceClaimDeviceStatus`, `DRADeviceBindingConditions`, and `DRAConsumableCapacity` feature gates have to be enabled in kube-apiserver.
- DP doesn't detect every device of a node, because the hardware the node itself depends on stays at its disposal. The following devices never appear in the list:
  - Display adapters, which are handled by the `gpu` module.
  - Devices integrated into the chipset, bridges, memory controllers, and system peripherals.

    - Network controllers with active interfaces.
    - Storage controllers whose disks are used by the node.
    - Devices that share an IOMMU group with any of the above, because a group goes to the VM as a whole.
- A VM starts only on the node that holds the device attached to it, so all its PCI devices have to be on the same node, otherwise the specification is rejected.
- A VM with a PCI device can't be moved by live migration, and the `Migratable` condition gets the `VirtualMachineHostDevicesNotMigratable` reason, so such machines have to be stopped before the node is put into maintenance.
- Devices are attached when the VM starts, so a change to the `.spec.pciDevices` parameter requires its restart.
- A single device is attached to one VM only, and a VM takes no more than eight devices.
- A network card that's passed through works around the cluster network subsystem. It doesn't appear in the `.spec.networks` parameter, and DP neither assigns nor tracks its IP and MAC addresses.

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Security

The "Security" section provides information about the protection mechanisms implemented in Deckhouse Platform.

It contains recommendations, instructions, and configuration examples for built-in protection mechanisms, as well as integration with external systems.

In this section, you will find information on:

- Security event audit:
    - How to enable and configure Kubernetes API event audit.
    - How to collect security events at the kernel and Kubernetes API levels using the platform's built-in capabilities (Falco).
    - How to configure audit rules and receive alerts on suspicious activity.
- Virtualization event audit:
    - How to enable the log of actions on virtual machines.
    - Which events it records and how to view them.
- Security policies:
    - Support for Pod Security Standards.
    - Configuring operational and advanced security policies using Gatekeeper.
    - Verifying container image signatures.
    - Working with custom policies and exceptions.
- Image vulnerability scanning:
    - How to set up regular scanning of container images.
    - How to view scan results and manually trigger rescans.
- Certificate management:
    - Issuing, renewing, and managing TLS certificates using the `cert-manager` module.
    - Examples of using Let's Encrypt, HashiCorp Vault, self-signed, and external CAs.
    - Support for `HTTP-01` and `DNS-01` validation types.
- Integration with external monitoring and security systems:
    - Sending logs to Kaspersky Unified Monitoring and Analysis Platform (KUMA).
    - Configuring exclusions for antivirus solutions, using Kaspersky Endpoint Security for Linux (KESL) as an example.

Additional resources

- Module `cert-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Security events

Deckhouse Platform (DP) provides tools for declarative collection, processing, normalization, and delivery of security events extracted from logs of applications and Kubernetes infrastructure components.

A security event is a structured record of an action or fact that is significant from an information security perspective.

DP lets you:

- Collect security events from various sources (pod logs, node files, Kubernetes API audit).
- Bring events to a unified format with a mandatory minimum set of attributes.
- Enrich events with contextual data.
- Filter events by sources and severity level.
- Deliver events to storage and analytics systems (Loki, Elasticsearch, Kafka, Splunk, Vector, and others).

The `security-events-manager` module is responsible for collecting, processing, and delivering security events. The auxiliary `log-shipper` module is used for log collection.

Dependencies and requirements

The `security-events-manager` module requires the following DP modules:

- `log-shipper`: Collects logs from pod sources and node files, performs preliminary record selection.
- `loki`: Provides in-cluster storage for security events (used by default as a destination).

For details about the security events architecture, refer to “Security events architecture”.

Data sources for security events

The `security-events-manager` module collects data from two types of sources:

- **Container sources**: Application logs in Kubernetes pods. Collection is performed via the `log-shipper` module, which selects records by pod labels and namespaces.
- **Cluster sources**: Node files and system service logs not bound to a specific namespace. For example:
  - `/var/log/kube-audit/audit.log`: Kubernetes API audit log.
  - `/var/log/auth.log`: Node system authentication event log.

Collection is made in two steps:

1. The `log-shipper` module performs preliminary selection of log records using simple comparison operators (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) and forwards them to the `security-events-manager` gateway.
2. The `security-events-manager` gateway parses the records, transforms them to a unified model, and delivers to a configured destination.

Since log parsing is resource-intensive, it is performed only for records pre-selected as potentially containing security events.

Enabling security events

To enable security events, follow these steps:

1. Enable the required modules if they are not yet enabled:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  enabled: true
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  enabled: true
```

2. Enable the `security-events-manager` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: security-events-manager
spec:
  enabled: true
```

All available `security-events-manager` parameters are listed in the module documentation.

Configuring event sources

Log collection and processing are configured through the following custom resources:

- PodSecurityEventShipper: For container sources (pod logs in a specific namespace).
- ClusterSecurityEventShipper: For cluster sources (node files, system service logs).

These resources configure:

- Log source (`source` and `input` fields)
- Preliminary record selection rules (`produces` field)
- Parsing rules (`parser` or `parserRef` field with a reference to SecurityEventLoggingTransformationRules or ClusterSecurityEventLoggingTransformationRules)
- Transformation and enrichment rules (`transform` and `enrich` fields)

PodSecurityEventShipper configuration example defining a container source:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: PodSecurityEventShipper
metadata:
  name: runtime-audit-engine
  namespace: d8-runtime-audit-engine
spec:
  - source: runtime-audit-engine
    input:
      type: KubernetesPods
      kubernetesPods:
        labelSelector:
          matchLabels:
            app: runtime-audit-engine
    produces:
      - eventCode: K8S_POD_CREATED
        extract:
          field: message
          operator: Regex
          values:
            - ".*K8s Pod Created.*"
```

ClusterSecurityEventShipper configuration example defining a cluster source:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventShipper
metadata:
  name: kube-audit
spec:
  - source: kube-audit
    input:
      type: File
      files:
        - /var/log/kube-audit/audit.log
    produces:
      - eventCode: UNAUTHORIZED_ACCESS
        extract:
          field: message
          operator: Regex
          values:
            - ".*\"code\":401.*"
```

Configuring event delivery

After events are formed, you should define the destinations they will be delivered to. For this, you need to:

1. Configure security event destinations.
2. Configure event delivery rules to destinations.

Configuring a destination

Destinations are configured via the ClusterSecurityEventDestination resource.

The `security-events-manager` module supports all destination types available in the `log-shipper` ecosystem (Loki, Elasticsearch, Kafka, Splunk, Vector, File, and others). For in-cluster security event storage, automatic Loki destination configuration is available when the corresponding module option is enabled.

ClusterSecurityEventDestination configuration example defining the Loki destination:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventDestination
metadata:
  name: cluster-loki
spec:
  type: Loki
  loki:
    auth:
      strategy: Bearer
      token: <EXAMPLE>
    endpoint: https://loki.d8-monitoring:3100
    tls:
      verifyCertificate: true
      verifyHostname: true
      ca: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...
```

Configuring event delivery rules

When configuring security event delivery rules, you need to define:

- Sources that will be sending the events
- Minimum severity level for sending
- Target destinations

For this, use the ClusterSecurityEventConfig resource. It defines sources (in the format of exact names or masks), minimum severity level, and an array of destinations (resources of type ClusterSecurityEventDestination) that will receive the selected events.

ClusterSecurityEventConfig configuration example for sending events from the `kube-audit`, `runtime-audit-engine` and `user-authn` sources to the `cluster-loki` destination:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventConfig
metadata:
  name: default
spec:
  defaultSeverityThreshold: Low
  enabledSources:
    - clusterSecurityEventShipper/kube-audit/kube-audit
    - podSecurityEventShipper/d8-runtime-audit-engine/runtime-audit-engine/falco
    - podSecurityEventShipper/d8-user-authn/user-authn/dex
  # Alternatively, use masks.
  # enabledSourcesMasks:
  #   - clusterSecurityEventShipper/kube-audit/*
  #   - podSecurityEventShipper/*

  destinations:
    - cluster-loki
```

Default module settings

If the module settings are not explicitly specified, the following objects are created in the cluster:

- ClusterSecurityEventConfig: Configures security event delivery to destinations. By default, an object with the following configuration is created:

  ```
  apiVersion: security.deckhouse.io/v1alpha1
  kind: ClusterSecurityEventConfig
  metadata:
    name: default
  spec:
    defaultSeverityThreshold: Medium
    destinations:
      - cluster-loki
    enabledSourcesMasks:
      - podSecurityEventShipper/*
      - clusterSecurityEventShipper/*
  ```

  This object configuration is controlled by the module parameter `securityEventConfig`.

- ClusterSecurityEventDestination: Configures security event destination. By default, the following object is generated that allows sending events to the in-cluster Loki service:

  ```
  apiVersion: security.deckhouse.io/v1alpha1
  kind: ClusterSecurityEventDestination
  metadata:
    name: cluster-loki
  spec:
    type: Loki
    loki:
      auth:
        strategy: Bearer
        token: <token> # Filled automatically.
      endpoint: https://loki.d8-monitoring:3100
  ```

  You can disable generation of the default destination via the dedicated parameter `clusterSecurityEventDestination.clusterLoki`.

Supported security events

The `security-events-manager` module ships with a built-in set of security event detection rules

covering authentication, configuration, RBAC, runtime, and other categories. For the up-to-date list of implemented security events, their codes, severity levels, and descriptions, refer to the module documentation.

Additional resources

- Module `log-shipper` documentation
- Module `loki` documentation
- Module `security-events-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Kubernetes API event audit

The Kubernetes auditing feature allows you to track requests to the API server and analyze events occurring in the cluster. Auditing can be useful for troubleshooting unexpected behavior and for meeting security requirements.

Kubernetes supports audit configuration via the Audit policy mechanism, which allows you to define logging rules for target operations. By default, audit results are written to the `/var/log/kube-audit/audit.log` file.

Built-in audit policies

Deckhouse Platform (DP) configures a basic audit policy by default, which can be supplemented with custom rules.

This policy implements the following rules:

- Do not log health probes (`/readyz`, `/livez`, `/healthz`). This rule does not apply to requests that fail authentication.
- Do not log frequent updates for `Endpoints`, `EndpointSlices`, and `Events`.
- Do not log leader election operations on `Lease` resources.
- Do not log cert-manager leader election ConfigMaps.
- Do not log VerticalPodAutoscalerCheckpoints resources.
- Do not log PATCH operations on `VerticalPodAutoscaler` from recommender.
- Do not log UpmeterHookProbes resources.
- Do not log any operations in `d8-upmeter` namespace.
- Do not log ingress-nginx leader election updates in ConfigMaps.
- Do not log dex health-check create/delete operations on `AuthRequest` resources.
- Log create and delete operations for Node resources with request/response payload.
- Log kubectl logs requests (pods/log) at Metadata level.
- Log create/update/patch/delete operations from system service accounts (`kube-system`, `d8-*`).
- Log create/update/patch/delete operations for Pod resources.
- Log Pod status updates from kubelet (pods/status) with the request body to capture container terminations and their reasons.
- Log create/update/patch/delete operations in system namespaces (`kube-system`, `d8-*`).
- Log all LIST operations in all namespaces.
- Log create and delete operations for ServiceAccount resources.
- Log create/update/delete/patch operations for Role and ClusterRole resources.
- Log create/update/delete operations for ClusterRoleBinding resources.
- Log exec, attach and ephemeral container related pod subresource operations.
- Log creation of VirtualMachineOperation resources with request/response payload.
- Log create/update/patch/delete operations for virtualization.deckhouse.io resources.
- Log update/patch operations for internal virtualization subresources.
- Log GET operations for subresources.virtualization.deckhouse.io API group.
- Log create/update/patch/delete operations for `Pod` resources.
- Log create/update/patch/delete operations in `d8-virtualization` namespace.
- Log create/update/patch/delete operations for ModuleConfig resources.
- Do not log requests from authenticated users.
- Log all remaining (unauthenticated) requests at Metadata level.

Detailed information about the rules and an example of the final policy are available in the control-plane-manager module documentation

These policies are enabled by default. These rules from the basic policy are enabled by default. To disable them, set the `basicAuditPolicyEnabled` parameter to `false`.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

Configuring a custom audit policy

To create an advanced Kubernetes API audit policy, follow these steps:

1. Enable the `auditPolicyEnabled` parameter in the `control-plane-manager` module configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 3
     settings:
       apiserver:
         auditPolicyEnabled: true
   ```

2. Create the `kube-system/audit-policy` Secret containing the policy YAML file encoded in Base64:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: audit-policy
     namespace: kube-system
   data:
     audit-policy.yaml: <Base64>
   ```

   Example `audit-policy.yaml` content with a minimal working configuration:

   ```
   apiVersion: audit.k8s.io/v1
   kind: Policy
   rules:
   - level: Metadata
     omitStages:
     - RequestReceived
   ```

   For more information on possible contents of `audit-policy.yaml`, refer to the following sources:

   - Kubernetes official documentation
   - GCE helper script code snippet

How to generate the contents of the `audit-policy.yaml` file

Kubernetes Documentation Policy Resource Field Structure

A Kubernetes audit policy is defined in a YAML file and consists of a set of rules that determine which events and at what level of detail will be logged. The file has the following structure:

```
apiVersion: audit.k8s.io/v1  # API version used for the audit policy
kind: Policy                # Resource type — constant Policy
rules:                       # Set of rules for auditing
  - level:                   # Log detail level. Required field
    users:                   # List of users whose actions are logged
    userGroups:               # User groups (e.g., system:serviceaccounts)
    verbs:                   # Actions/operations that are logged (create, update,
delete, etc.)
    resources:                # Kubernetes resources for which the rule applies
    namespaces:               # Namespaces that the rule covers
```

The `rules` array describes the audit rules. Each rule contains the following fields:

- `level` — The detail level of the logged event.

  Possible values (from most to least detailed):

  - `None`: Do not log at all.
  - `Metadata`: Only request metadata (who, when, what, where; without object contents).
  - `Request`: Also stores the request body (for update requests only).
  - `RequestResponse`: Stores both the request body and the response content.

- `users`: A list of `usernames` the rule applies to (e.g., `["admin"]`). For service accounts, the name typically looks like `system:serviceaccount:<namespace>:<serviceaccount-name>`. For regular users, the name depends on the authentication system settings. For `deckhouse.io/v1` `Users` objects, `email` is used as the name.

- `userGroups`: `userGroups` (e.g., `["system:authenticated"]`). After authentication, kube-apiserver assigns a list of groups to each user (e.g., all authenticated users are part of `system:authenticated`, service accounts are in additional groups). If a request comes from a user who is a member of at least one of the groups specified in `userGroups`, the rule is applied to that request.

  Kubernetes built-in groups:

  - `system:authenticated`: Everyone who has been authenticated.
  - `system:unauthenticated`: Requests from anonymous users.
  - `system:serviceaccounts`: All service accounts from all namespaces.
  - `system:serviceaccounts:<namespace>`: Service accounts in a specific namespace.

- `verbs`: List of API operations (get, list, create, delete, etc.).

- `resources`: Array of target resources:
  - `group`: API group (e.g., `"apps"`, `"batch"`, `""` for core).
  - `resources`: Resource types (e.g., `["pods", "deployments"]`). A full list of resources and their groups can be obtained using the `kubectl api-resources` command

- `namespaces`: Array of namespaces where the rule applies.

- `nonResourceURLs`: A set of URL paths to audit. The * character is allowed, but only as a full, final step of the path. Examples:
  - `/metrics` — log requests to apiserver metrics
  - `/healthz*` — log all health requests

Examples

Log all requests from all authenticated users

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
```

⚠ **Warning!** Not recommended for production environments. A high event flow may result in increased load on the control-plane disk subsystem of cluster nodes.

Example log entry:

```
{
  "stage": "ResponseComplete",
  "requestURI": "/healthz",
  "verb": "get",
  "user": {
    "username": "kube-apiserver-kubelet-client",
    "groups": [
      "kubeadm:cluster-admins",
      "system:authenticated"    //  <- everyone who has been authenticated
    ]
  },
  ...
}
```

Logging all requests not from service accounts

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
    users: []
    omitStages: []
    # Exclude service accounts via a deny rule
  - level: None
    userGroups: ["system:serviceaccounts"]
```

Example log entry:

```
{
  "user": {
    "username": "user@example.com",
    "groups": ["admins","system:authenticated"]
  },
  ...
}
```

Logging all operations with standard resources

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    resources:
      - group: "apps"
        resources: ["deployments", "daemonsets", "statefulsets", "replicasets"]
      - group: "batch"
        resources: ["jobs", "cronjobs"]
      - group: ""
        resources: ["*"]
```

Working with the audit log file

On DP master nodes, it is assumed that a log collection tool (`log-shipper`, `promtail`, or `filebeat`) is installed to monitor the `/var/log/kube-audit/audit.log` file.

The log rotation settings for this file are predefined and cannot be changed:

- Maximum file size: 1000 MB.

- Maximum retention period: 30 days.

Depending on the policy configuration and the volume of requests to the API server, the number of log entries can be very large. In such cases, the retention period may be reduced to less than 30 minutes.

⚠ Unsupported options or typos in the configuration file may cause the API server to fail to start.

If the API server fails to start, take the following steps:

1. Manually remove the `--audit-log-*` parameters from the `/etc/kubernetes/manifests/kube-apiserver.yaml` manifest.
2. Restart the API server with the following command:

```
docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')

# Alternative option (depending on the CRI in use).
crictl stopp $(crictl pods --name=kube-apiserver -q)
```

3. After restarting, fix the Secret or delete it with the following command:

```
d8 k -n kube-system delete secret audit-policy
```

Redirecting the audit log file to stdout

By default, the audit log is saved to the `/var/log/kube-audit/audit.log` file on master nodes. If necessary, you can redirect its output to the `kube-apiserver` process stdout instead of a file by setting the `apiserver.auditLog.output` parameter in the `control-plane-manager` module to `Stdout`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

In this case, the log will be available in the `kube-apiserver` container stdout.

Then, using the built-in DP logging mechanism , you can configure log collection and forwarding to your own security system.

Additional resources

- Module `control-plane-manager` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Runtime audit

Deckhouse Platform (DP) provides built-in tools for detecting security threats by analyzing Linux kernel events and auditing Kubernetes API events. With DP, you can:

- Detect threats in environments by analyzing applications and containers.
- Identify attempts to exploit vulnerabilities from the CVE database and signs of cryptocurrency miner activity.
- Detect Kubernetes-specific threats, including:
  - Command-line shells launched in containers or pods.
  - Containers running in privileged mode.
  - Mounting insecure paths into containers (for example, `/proc`).
  - Attempts to read sensitive data (for example, from `/etc/shadow`).

Data sources for runtime audit

DP uses two main sources of events:

- Linux kernel events — via the eBPF driver for the Falco threat detection system.
- Kubernetes API audit events — via integration with Kubernetes auditing and a webhook interface.

For details about the runtime audit architecture, refer to “Runtime audit architecture”.

Minimum requirements

To capture kernel events, you need:

- Linux kernel version 5.8 or higher.
- Support for eBPF. You can check for support in one of the following ways:
  - Make sure the `/sys/kernel/btf/vmlinux` file exists:

    ```
    ls -lah /sys/kernel/btf/vmlinux
    ```

  - Make sure the `CONFIG_DEBUG_INFO_BTF` option is enabled:

    ```
    grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*
    ```

Falco agents run on each cluster node and consume resources depending on the number of rules applied and events collected.

> **i** On some systems, eBPF probes may not work.

Enabling runtime audit

To enable the runtime audit, follow these steps:

1. Make sure the nodes meet the minimum requirements.
2. Enable the `runtime-audit-engine` module using the following configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: runtime-audit-engine
   spec:
     enabled: true
   ```

3. (**Optional**) If the cluster control plane is not managed by DP with `control-plane-manager`, configure the Kubernetes API audit webhook manually.

All available runtime audit parameters are listed in the `runtime-audit-engine` module documentation.

Manually configuring the Kubernetes API audit webhook

> **i** Webhook configuration is not required if the `control-plane-manager` module is enabled. In this case, when the `runtime-audit-engine` module is enabled, the settings for collecting Kubernetes API audit events will be applied automatically.

To configure the webhook for receiving audit events from `kube-apiserver`:

1. Create a `kubeconfig` file for the webhook with the address `https://127.0.0.1:9765/k8s-audit` and certificate data (`ca.crt`) from the `d8-runtime-audit-engine/runtime-audit-engine-webhook-tls` Secret.

   Example:

   ```
   apiVersion: v1
   kind: Config
   clusters:
     - name: webhook
       cluster:
         certificate-authority-data: BASE64_CA
         server: "https://127.0.0.1:9765/k8s-audit"
   users:
     - name: webhook
   contexts:
     - context:
         cluster: webhook
         user: webhook
       name: webhook
   current-context: webhook
   ```

2. Specify the path to the configuration file with the `--audit-webhook-config-file` flag in the `kube-apiserver` manifest.
3. (**Optional**) To collect Kubernetes API audit events from not only system but also user namespaces, configure audit policies.

Working with audit rules

Runtime audit event analysis is based on rules that define suspicious behavior criteria. DP includes:

- **Built-in rules**, including:
  - Kubernetes audit rules (located in the `falco` container at `/etc/falco/k8s_audit_rules.yaml`).

  To configure the list of built-in rules, use the `settings.builtInRulesList` parameter of the `runtime-audit-engine` module.

- **Custom rules**, defined via the FalcoAuditRules custom resource.

For more information on how runtime audit rules work, refer to “Runtime audit architecture”.

Adding a custom rule

To add a rule, create a FalcoAuditRules resource with the required conditions. Use the Falco condition syntax. Falco agents will automatically apply the created rule.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: ownership-permissions
spec:
  rules:
    - macro:
        name: spawned_process
        condition: (evt.type in (execve, execveat) and evt.dir=<)
    - rule:
        name: Detect Ownership Change
        desc: detect file permission/ownership change
        condition: >
          spawned_process and proc.name in (chmod, chown) and proc.args contains
"/tmp/"
        output: >
          The file or directory below has had its permissions or ownership changed
(user=%user.name
          command=%proc.cmdline file=%fd.name parent=%proc.pname
pcmdline=%proc.pcmdline gparent=%proc.aname[2])
        priority: Warning
        tags: [filesystem]
```

For more rule examples, see:

- Falco rules official repository
- Falco rules on Artifact Hub

Applying a third-party rule

Since Falco rule structure differs from DP custom resource format, third-party rules from the internet must be converted to a FalcoAuditRules resource before applying them.

Use the following script to convert:

```
git clone github.com/deckhouse/deckhouse
cd deckhouse/ee/modules/runtime-audit-engine/hack/far-converter
go run main.go -input /path/to/falco/rule_example.yaml > ./my-rules-cr.yaml
```

Example conversion result:

- Original rule:

  ```
  # /path/to/falco/rule_example.yaml
  - macro: spawned_process
     condition: (evt.type in (execve, execveat) and evt.dir=<)

  - rule: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
     desc: "This rule detects an attempt to exploit a container escape
  vulnerability in the Linux Kernel."
     condition: container.id != "" and proc.name = "unshare" and spawned_process
  and evt.args contains "mount" and evt.args contains "-o rdma" and evt.args
  contains "/release_agent"
     output: "Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
  (user=%user.loginname uid=%user.loginuid command=%proc.cmdline
  args=%proc.args)"
     priority: CRITICAL
     tags: [process, mitre_privilege_escalation]
  ```

- Converted resource:

```
# ./my-rules-cr.yaml
apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: rule-example
spec:
  rules:
    - macro:
        name: spawned_process
        condition: (evt.type in (execve, execveat) and evt.dir=<)
    - rule:
        name: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
        condition: container.id != "" and proc.name = "unshare" and
spawned_process and evt.args contains "mount" and evt.args contains "-o rdma"
and evt.args contains "/release_agent"
        desc: This rule detects an attempt to exploit a container escape
vulnerability in the Linux Kernel.
        output: Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-
0492) (user=%user.loginname uid=%user.loginuid command=%proc.cmdline
args=%proc.args)
        priority: Critical
        tags:
          - process
          - mitre_privilege_escalation
```

Log collection and alerts

DP exports runtime audit events as Prometheus metrics, allowing you to set up log collection and alerts via resources of the `log-shipper` and `observability` modules. This makes it possible to:

- Connect an external log storage (for example, Loki or Elasticsearch).
- Configure alerts for critical events.

Configuring log and event collection

All runtime audit events are sent to stdout. To collect and forward events to a log storage, create a ClusterLoggingConfig resource following the example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: falco-events
spec:
  destinationRefs:
  - xxxx
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
        - key: "kubernetes.io/metadata.name"
          operator: In
          values: [d8-runtime-audit-engine]
  labelFilter:
  - operator: Regex
    values: ["\\{.*"] # Collect only JSON-formatted logs.
    field: "message"
  type: KubernetesPods
```

Configuring critical event alerts

To create alerts for critical events:

1. Make sure the `observability` module is enabled.
2. Create a ClusterObservabilityMetricsRulesGroup or ObservabilityMetricsRulesGroup resource:
   - ClusterObservabilityMetricsRulesGroup: Intended for cluster-wide rules applied to system namespaces.
   - ObservabilityMetricsRulesGroup: Intended for rules that operate within a specific namespace.

Configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: falco-critical-alerts
spec:
  groups:
    - name: falco-critical-alerts
      rules:
        - alert: FalcoCriticalAlertsAreFiring
          for: 1m
          annotations:
            description: |
              There is a suspicious activity on a node {{ $labels.node }}.
              Check your events journal for more details.
            summary: Falco detected a critical security incident.
          expr: |
            sum by (node)
(rate(falcosecurity_falcosidekick_falco_events_total{priority_raw="critical"}[5m]) >
0)
```

Viewing metrics

To retrieve Prometheus metrics, use the PromQL query `falcosecurity_falcosidekick_falco_events_total{}`:

```
d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
  curl -s "http://127.0.0.1:9090/api/v1/query?
query=falcosecurity_falcosidekick_falco_events_total" | jq
```

The event severity is carried by two labels: `priority` holds a numeric value, and `priority_raw` holds the lowercase severity name. Use `priority_raw` to filter events by severity:

priority	priority_raw
`8`	`emergency`
`7`	`alert`
`6`	`critical`
`5`	`error`
`4`	`warning`
`3`	`notice`
`2`	`informational`
`1`	`debug`

Debugging and simulating events

For debugging and simulating runtime audit events in DP, you can use:

- The `event-generator` utility.
- The `/test` HTTP endpoint of the `falcosidekick` service.

Enabling debug logging

Falco uses the `debug` logging level by default.

In Falcosidekick, debug logging is disabled by default. To enable it, set the `spec.settings.debugLogging` parameter to `true`, for example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: runtime-audit-engine
spec:
  version: 1
  enabled: true
  settings:
    debugLogging: true
```

Simulating events

Falco

The `event-generator` utility allows you to generate various suspicious actions (for example, system calls or Kubernetes API audit events).

Run the following command to generate a test set of events in a Kubernetes cluster:

```
d8 k run falco-event-generator --image=falcosecurity/event-generator run
```

If you need to simulate a specific action, refer to the utility's documentation.

Falcosidekick

To simulate sending test events to the `falcosidekick` service, use its `/test` HTTP endpoint:

1. Create a test event by running:

   ```
   nsenter -t $(pidof falcosidekick) curl -X POST -H "Content-Type: application/json" -H
   "Accept: application/json" http://localhost:2801/test
   ```

2. Check the event metric:

   ```
   d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
     curl -s "http://127.0.0.1:9090/api/v1/query?
   query=falcosecurity_falcosidekick_falco_events_total" \
     | jq '.data.result.[] | select (.metric.priority_raw == "debug")'
   ```

   Example output:

```
{
  "metric": {
    "__name__": "falcosecurity_falcosidekick_falco_events_total",
    "container": "kube-rbac-proxy",
    "hostname": "falcosidekick",
    "instance": "192.168.208.7:4212",
    "job": "runtime-audit-engine",
    "node": "dev-master-0",
    "priority": "1",
    "priority_raw": "debug",
    "rule": "Test rule",
    "source": "internal",
    "tier": "cluster"
  },
  "value": [
    1744234729.799,
    "1"
  ]
}
```


Additional resources

- Module `control-plane-manager` documentation
- Module `log-shipper` documentation
- Module `observability` documentation
- Module `runtime-audit-engine` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtualization event audit

The audit records actions on virtual machines (VMs) and on the module itself, so that you can investigate an incident and reconstruct the sequence of events.

Enabling the audit

To enable the security event audit, follow these steps:

1. Enable the `log-shipper` and `runtime-audit-engine` modules.
2. Enable the Kubernetes API audit by setting `.spec.settings.apiserver.auditPolicyEnabled` to `true` in the `control-plane-manager` module.
3. Set `.spec.settings.audit.enabled` to `true` in the module settings:

```
spec:
  settings:
    audit:
      enabled: true
```

Until all three conditions are met, the audit component doesn’t start in the cluster. For the other parameters, see the module settings.

Event types

The event type is recorded in the `type` field. The audit distinguishes the following types:

- `Access to VM`: A connection to a VM over the console, VNC, or port forwarding. Both the start and the end of the session are recorded.
- `Manage VM`: Creating, updating, or deleting a VirtualMachine resource.

- `Control VM`: A change of the VM state, including start, stop, restart, migration, and eviction through the VirtualMachineOperation resource, as well as a shutdown or restart from the guest OS and an abnormal termination.
- `Module control`: Creating, updating, disabling, or deleting a ModuleConfig.
- `Virtualization control`: Creating or deleting a system component of the module in the `d8-virtualization` namespace.
- `Integrity check`: A mismatch between the VM configuration checksum and the reference one.
- `Forbidden operation`: An attempt to perform a forbidden operation.

Regardless of the type, every event contains the same fields:

- `name`: A description of what happened.
- `datetime`: The time of the event.
- `request_subject`: The user or ServiceAccount that performed the action.
- `operation_result`: The result of the operation.
- `uid`: The identifier of the record in the Kubernetes audit.

Additional fields depend on the event type. For example, VM events contain the `virtual_machine_name` and `virtual_machine_namespace` fields, while forbidden operations report the request source in the `source_ip` field and the denial reason in the `forbid_reason` field.

Viewing events

Events are collected by the `virtualization-audit` system component in the `d8-virtualization` namespace. To forward them to the cluster logging system, for example to Loki, create a ClusterLoggingConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: virtualization-audit-logs
spec:
  destinationRefs:
    - d8-loki
  kubernetesPods:
    namespaceSelector:
      matchNames:
        - d8-virtualization
    labelSelector:
      matchLabels:
        app: virtualization-audit
  type: KubernetesPods
```

To view the events in Grafana, use a Loki query:

```
{namespace="d8-virtualization", pod=~"virtualization-audit-.*"}
```

Additional resources

- Global custom resources
- Module `control-plane-manager` documentation
- Module `log-shipper` documentation
- Module `runtime-audit-engine` documentation
- Module `virtualization` parameters
- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Security policies

Deckhouse Platform (DP) lets you manage application security in the cluster using a set of admission

policies. These are rules that apply to objects (such as Pods and Services) at the time of their creation and modification in the cluster (but not during their operation), based on the information provided in their manifests. These policies are designed to formalize the parameters that are permitted or prohibited in object manifests. Support for admission policies in the DP cluster is implemented using the `admission-policy-engine` module.

In the DP policies are divided into three categories:

- Pod Security Standards: Policies that comply with the relevant Pod Security Standards.
- Operational policies: Policies for creating additional requirements for objects by validating the values of parameters that are **not directly related** to security (for example, a list of allowed prefixes for container images, an image download policy, a list of required container images, etc.).
- Security policies: Policies for creating additional requirements on objects by validating the values of security-related parameters (for example, container access to the host's IPC or PID namespaces, privilege lists for containers, etc.).

> **i** These policies complement each other. If multiple policies are applied to a single namespace, objects are validated against each of them. If even one policy is violated, the object will not be created.

In addition to policies that prohibit using parameters different from the set requirements, DP supports the SecurityPolicyException resource, which allows creating fine-grained exceptions from security policy checks. With this resource, you can allow using specific parameters for individual pods or containers without changing security policies applied to the entire namespace.

How validation failure messages are displayed

Depending on how pods are created, there are differences in how the API generates messages regarding validation failures (violations of established policies):

- If a pod is created directly, the validation error is returned in the API response indicating a validation failure (policy violation).
- If pods are created via Deployment, the required number of ReplicaSets is created, which in turn attempt to create the pods. In this case, the validation error is not returned in the API response but is displayed in the namespace events or the corresponding ReplicaSet events.

Pod validation when policies are modified or added

For all three policy categories (Pod Security Standards, operational, and security policies), there is no provision for automatically recreating existing pods when changing existing policies or adding new ones. Pods that existed prior to changes being made to the policy in use or prior to a new policy being added will continue to run until they are restarted. Upon restart, they will be validated against the new rules.

DP provides alerts for such cases. Each alert fires once for each policy and namespace with violating objects:

Alert	What it reports
`PodSecurityStandardsViolation`	Running pods that violate a Pod Security Standard
`OperationPolicyViolation`	Existing objects that violate an OperationPolicy
`SecurityPolicyViolation`	Existing objects that violate a SecurityPolicy
`ControllerPolicyViolation`	Pod-creating controllers that violate a policy with `enforcementAction: Deny` and are only warned about during the transition period

The `policy` label of an alert names the policy: the name of the OperationPolicy or SecurityPolicy, or `d8-pod-security-baseline` or `d8-pod-security-restricted` for the Pod Security Standards. The `namespace` label is the namespace of the violating objects.

An alert is an ObservabilityAlert resource in the namespace of the violating objects, so the users of the namespace see it. An alert for cluster-scoped objects is a ClusterObservabilityAlert resource.

To list the alerts in a namespace, run the following command, where `<NAMESPACE>` is the namespace:

```
d8 k -n <NAMESPACE> get observabilityalerts
```

Output example:

```
NAME                                                        SEVERITY   STATUS
DURATION   SUMMARY
AGE
PodSecurityStandardsViolation-07243a74fddd8234-1792744609   3          Firing   1h
Pods in the `app` namespace violate the `d8-pod-security-baseline` Pod Security
Standard.   1h
SecurityPolicyViolation-8f21cd962274a252-1792744609         3          Firing   1h
Objects in the `app` namespace violate the `restricted-app` SecurityPolicy.
1h
```

To list the alerts in every namespace and the alerts for cluster-scoped objects, run the following commands:

```
d8 k get observabilityalerts -A
d8 k get clusterobservabilityalerts
```

To view an alert, run the following command:

```
d8 k -n <NAMESPACE> get observabilityalert <ALERT_NAME> -o yaml
```

The description of the alert contains a Prometheus query that lists the violating objects and the rules they break.

Applying Pod Security Standards

DP supports three security policy levels:

- `privileged`: An unrestricted policy with the broadest possible permissions.
- `baseline`: A minimally restrictive policy that prevents the most well-known and common privilege escalation techniques. Allows the use of a standard (minimally specified) Pod configuration.
- `restricted`: A highly restrictive policy with the strictest requirements for Pods.

> **i** In the Deckhouse Platform, these policies are implemented using Gatekeeper and enforced by the admission controllers of the `admission-policy-engine` module, rather than the Kubernetes Pod Security Admission controller. Only the policy descriptions are taken from Kubernetes.

Default policy

The default policy is `baseline`. It applies to non-system namespaces without the `security.deckhouse.io/pod-policy` label.

> **⚠** In DP versions prior to v1.78, the `privileged` policy was applied by default in clusters bootstrapped with DP prior to v1.55 or when the bootstrap version could not be determined. Starting from DP v1.78, the default policy is `baseline` regardless of the cluster bootstrap version.
>
> If the `enforcementAction` parameter is set to `Deny`, pods that do not comply with the `baseline` policy are denied when recreated. Updates of such clusters to DP v1.78 are blocked until the default policy is set explicitly.
>
> If the `admission-policy-engine` module is disabled, the update is not blocked. After the module is enabled, the `baseline` policy is applied.
>
> To keep the `privileged` policy, set it explicitly in the `settings.podSecurityStandards.defaultPolicy` parameter.

Assigning a policy

You can assign a policy in the following ways:

- Globally, using the `settings.podSecurityStandards.defaultPolicy` parameter of the `admission-policy-engine` module.

- Per namespace, using the `security.deckhouse.io/pod-policy=<POLICY_NAME>` label.

  Example command to assign the `restricted` policy to all Pods in the `my-namespace` namespace:

  ```
  d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
  ```

Enforcement modes

Allowed policy enforcement modes:

- `deny`: Blocks actions from being executed.
- `dryrun`: Does not affect execution and used for debugging. Event information can be viewed in the Deckhouse web UI or with `d8 k`.
- `warn`: Works like `dryrun` but also displays a warning with the reason the action would have been denied in `deny` mode.

By default, Pod Security Standards policies in DP are enforced in `deny` mode. In this mode, application Pods that do not comply with the policies cannot be run in the cluster.

As with policy assignment, enforcement mode can be set:

- Globally, using the `settings.podSecurityStandards.enforcementAction` parameter of the `admission-policy-engine` module.
- Per namespace, using the `security.deckhouse.io/pod-policy-action=<POLICY_ACTION>` label.

  Example command to set the `warn` mode for all Pods in the `my-namespace` namespace:

  ```
  d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
  ```

Policies in system namespaces

Namespaces named `d8-*` and `kube-*` hold the components of the platform itself. Policies apply to them differently from application namespaces, and that difference is not configurable. The platform also labels the namespaces it creates with `heritage: deckhouse`. A namespace that carries the label but is named otherwise is left out of policies and mutations written for application namespaces; the Pod Security Standards below follow the names.

Every namespace named `d8-*` or `kube-*` is checked against the `restricted` standard. The `security.deckhouse.io/pod-policy` label and the `settings.podSecurityStandards.defaultPolicy` parameter do not apply there. A violation is recorded in the audit and shown in the Deckhouse web UI, and the workload still starts. The exception is a namespace whose module has hardened it: there the standards are enforced and a violating workload is denied.

These checks cannot be tuned from outside the platform. The labels that govern them, and the workloads they cover, belong to the module that owns the namespace, and Deckhouse Platform returns both to their declared state the next time it applies the configuration. A module exempts its own workloads where necessary with a SecurityPolicyException that it ships with the module.

OperationPolicy and SecurityPolicy resources also apply to system namespaces in `warn` mode. A policy with `enforcementAction: Deny` blocks workloads in application namespaces and only reports violations in a system namespace. No label of the namespace changes that: a module that hardens its own namespace raises the Pod Security Standards there, which has no effect on a policy written for application workloads.

A denying policy that reaches system namespaces is therefore rendered as two Gatekeeper constraints, both visible in the audit and in the web interface:

- The policy’s own name: For application namespaces, with the action the policy asks for.
- `d8-system-default-<policy>`: For system namespaces, in `warn` mode.

For the transition to controller-level validation, a denying policy only warns about pod-creating controllers by default. It checks them in one more constraint, `d8-controllers-<policy>`, in `warn` mode, and the constraint with the policy’s own name checks Pods only. The `podSecurityStandards.controllerValidationEnforcement: Policy` module parameter, or `spec.controllerValidation.mode: Enforce` of the policy, checks controllers with the action of the policy in the single constraint instead.

The `d8-system-default-`, `d8-pod-security-` and `d8-controllers-` prefixes are reserved: a policy whose name starts with one of them is rejected on creation. A policy name is limited to 235 characters

for the same reason, so that the derived constraint names stay within the 253-character limit for a Kubernetes object name.

A policy is rendered as a single constraint when the split would change nothing:

- The policy uses the `Warn` or the `Dryrun` action.
- The namespaces the policy selects include no system namespace.
- The policy already excludes every system namespace it selects.
- The namespace list uses a leading glob, such as `*-system`, which cannot be intersected with `d8-*` exactly. The single constraint then keeps the policy's action and excludes system namespaces outright.
- The policy selects system namespaces only, in which case the single constraint keeps the policy's name and warns.

Gatekeeper mutations do not apply in system namespaces, whatever labels the namespace carries. The platform sets the parameters of its own components, so an `Assign` or a `ModifySet` resource is not allowed to change them there.

Extending a policy

You can extend the `baseline` and `restricted` policies using Gatekeeper templates by adding extra checks to the existing ones.

To extend a policy:

1. Create a validation template using a `ConstraintTemplate`.
2. Apply the template to the `baseline` or `restricted` policy.

Example template for validating the container registry address:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }
```

Example of applying the template to the `restricted` policy:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"
```

In this example, the image address in the `image` field of all Pods in namespaces labeled `security.deckhouse.io/pod-policy: restricted` is checked. If the `image` address of a created Pod does not start with `mycompany.registry.com`, the Pod will not be created.

Helpful resources for creating extended policies:

- Custom policy examples.
- Gatekeeper documentation contains information on templates and policy language.
- Gatekeeper library contains examples of validation templates.

Operational policies

DP provides a mechanism for creating operational policies using the OperationPolicy. Operational policies define requirements for cluster objects. These include allowed container registries, required resources, probes, and more.

The DP development team recommends applying the following minimal operational policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.io
    requiredResources:
      limits:
        - memory
      requests:
        - cpu
        - memory
    disallowedImageTags:
      - latest
    requiredProbes:
      - livenessProbe
      - readinessProbe
    maxRevisionHistoryLimit: 3
    imagePullPolicy: Always
    priorityClassNames:
    - production-high
    - production-low
    checkHostNetworkDNSPolicy: true
    checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/enabled: "true"
```

This policy defines basic operational requirements for cluster objects, including allowed container registries, required resources and probes, restrictions on using images with the `latest` tag, allowed priority classes, and other settings that improve application security and stability.

To assign this operational policy, add the `operation-policy.deckhouse.io/enabled=true` label to the target namespace:

```
d8 k label ns my-namespace operation-policy.deckhouse.io/enabled=true
```

Security policies

Using the SecurityPolicy, you can create security policies that define container behavior restrictions in the cluster, such as host network access, privileges, AppArmor usage, and more.

> **i** Detailed information on pod and container security settings (such as `hostNetwork`, `hostPID`, `hostIPC` and others), available values, and practical suggestions are available on the “Pod and container security settings” page.

Example security policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
    - driver: vmware
    allowedHostPorts:
    - max: 4000
      min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
    - unconfined
    allowedUnsafeSysctls:
    - kernel.*
    allowedVolumes:
    - hostPath
    - projected
    fsGroup:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - ALL
    runAsGroup:
      ranges:
      - max: 500
        min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
      - my_profile.json
      allowedProfiles:
      - Localhost
    supplementalGroups:
      ranges:
      - max: 133
        min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          enforce: mypolicy
```

To assign this security policy, add the `enforce: "mypolicy"` label to the target namespace.

Security policy exceptions

SecurityPolicyException is a resource that lets you create fine-grained exceptions from security policy checks for individual pods and containers. It allows you to avoid excluding an entire namespace from checks and instead define only the necessary exceptions from a specific rule for a pod or container.

Adding exceptions

To add exceptions for a pod or container, do the following:

1. Create a SecurityPolicyException object describing the required exceptions.

   It is recommended that you describe the reason for each exception in the rule's `metadata` field (for example, `metadata.description`). This makes auditing and maintenance easier.

2. In the pod template (usually via `spec.template.metadata.labels` in a Deployment, StatefulSet, or DaemonSet resource), add one of the following labels referencing the exception:

   - `security.deckhouse.io/security-policy-exception: <exception-name>`: Exception for the entire pod.
   - `security.deckhouse.io/security-policy-exception.container.<container-name>: <exception-name>`: Exception for a specific container.

Priority when selecting an exception for a container:

1. The label `security.deckhouse.io/security-policy-exception.container.<container-name>` is checked first.
2. If the container-specific label is absent, the exception from `security.deckhouse.io/security-policy-exception` is used.

⚠ If a container-specific label is set for a container but it points to an invalid or non-existent SecurityPolicyException object, it still has priority over the global label and may lead to pod placement denial.

Configuration example

For this example, consider a pod that requires:

- Permission to use the `hostNetwork` parameter for the entire pod.
- Permission to use the `privileged` parameter only for the `sample-init` container.

Without the SecurityPolicyException resource, allowing these parameters would require implementing a custom security policy where these settings could be allowed for any pod in the cluster.

With SecurityPolicyException, it is enough to create only the following resources:

- Exception to allow the `hostNetwork` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicyException
metadata:
  name: allow-hostnetwork-pod
spec:
  network:
    hostNetwork:
      allowedValue: true
      metadata:
        description: >-
          Pod requires host network mode for node-level network diagnostics.
```

- Exception to allow the `privileged` parameter in the `sample-init` container:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicyException
metadata:
  name: allow-privileged-init-container
spec:
  securityContext:
    privileged:
      allowedValue: true
      metadata:
        description: >-
          Container init requires privileged mode to access host-level networking
features.
```

After that, the corresponding labels need to be added to the pod template:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example
spec:
  template:
    metadata:
      labels:
        # General exception applicable to the entire pod.
        security.deckhouse.io/security-policy-exception: allow-hostnetwork-pod
        # Exception applicable to the sample-init container.
        security.deckhouse.io/security-policy-exception.container.sample-init: allow-
privileged-init-container
    spec:
      hostNetwork: true
      # Other pod configuration.
    containers:
      - name: sample-init
        securityContext:
          privileged: true
```

Partial policy enforcement

To enforce specific security policies without disabling the entire predefined set, follow these steps:

1. Add the `security.deckhouse.io/pod-policy: privileged` label to the target namespace to disable the built-in policy set.
2. Create a SecurityPolicy that matches the `baseline` or `restricted` level. In the `policies` section, specify only the security settings you need.
3. Add an extra label to the namespace matching the `namespaceSelector` in the SecurityPolicy.

Example SecurityPolicy configuration for the `baseline` level:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
```

```
    - localhost/*
  allowedCapabilities:
    - AUDIT_WRITE
    - CHOWN
    - DAC_OVERRIDE
    - FOWNER
    - FSETID
    - KILL
    - MKNOD
    - NET_BIND_SERVICE
    - SETFCAP
    - SETGID
    - SETPCAP
    - SETUID
    - SYS_CHROOT
  allowedHostPaths: []
  allowedHostPorts:
    - max: 0
      min: 0
  allowedProcMount: Default
  allowedUnsafeSysctls:
    - kernel.shm_rmid_forced
    - net.ipv4.ip_local_port_range
    - net.ipv4.ip_unprivileged_port_start
    - net.ipv4.tcp_syncookies
    - net.ipv4.ping_group_range
    - net.ipv4.ip_local_reserved_ports
    - net.ipv4.tcp_keepalive_time
    - net.ipv4.tcp_fin_timeout
    - net.ipv4.tcp_keepalive_intvl
    - net.ipv4.tcp_keepalive_probes
  seLinux:
    - type: ""
    - type: container_t
    - type: container_init_t
    - type: container_kvm_t
    - type: container_engine_t
  seccompProfiles:
    allowedProfiles:
      - RuntimeDefault
      - Localhost
      - undefined
      - ''
    allowedLocalhostFiles:
      - '*'
 match:
  namespaceSelector:
    labelSelector:
      matchLabels:
        operation-policy.deckhouse.io/baseline-enabled: "true"
```

Example SecurityPolicy configuration for the `restricted` level:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
 name: restricted
spec:
 enforcementAction: Deny
 policies:
  allowHostIPC: false
  allowHostNetwork: false
```

```
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
    requiredDropCapabilities:
      - ALL
    runAsUser:
      rule: MustRunAsNonRoot
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/restricted-enabled: "true"
```

Gatekeeper custom resources

Gatekeeper offers advanced capabilities for modifying Kubernetes resources using mutation policies. These policies are defined through the following custom resources:

- AssignMetadata: Modifies the `metadata` section of a resource.
- Assign: Modifies fields other than `metadata`.
- ModifySet: Adds or removes values from a list, such as container run arguments.

- AssignImage: Modifies the `image` field of a resource.

For more on modifying Kubernetes resources using mutation policies, refer to the Gatekeeper documentation.

Custom Gatekeeper policy examples

Here you can find examples of Gatekeeper policies that let you extend the standard cluster security mechanism.

Preventing deletion of a node without a specified label

> **i** The `DELETE` operations are handled by Gatekeeper by default.

You can create a Gatekeeper policy to prevent a node deletion unless it has a special label assigned.

In the following example, the `oldObject` field is used to check labels on the node being deleted:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Node deletion is blocked. Add label %q=%q to proceed.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"
```

Preventing exec and attach operations to specific Pods

The `admission-policy-engine` module's webhook routes `CONNECT` requests for `pods/exec` and `pods/attach` in namespaces other than `d8-*` and `kube-*` through Gatekeeper. This allows custom policies to allow or deny `kubectl exec` and `kubectl attach` operations.

Pods labeled heritage: deckhouse

The `deckhouse` module denies the following operations into pods of system components labeled `heritage: deckhouse` for all users, but the Deckhouse Platform system users:

- `kubectl exec`
- `kubectl attach`
- `kubectl cp`
- `kubectl port-forward`
- `kubectl debug`

The operations are denied regardless of whether the `admission-policy-engine` module is enabled in the cluster. For details, refer to the FAQ of the `deckhouse` module.

Custom policy example

You can create your own Gatekeeper policy to deny `kubectl exec` and `kubectl attach` operations in specific namespaces. In the following example, `input.review.operation` and `input.review.resource.resource` are used to check for `CONNECT` operations:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource is preferred, but fall back to subResource for older APIs
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource",
input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
        }
```

```
      violation[{"msg": msg}] {
        is_connect
        is_exec_or_attach
        is_forbidden_namespace
        msg := sprintf("Exec/attach is forbidden in namespace %q",
[input.review.namespace])
      }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
  name: deny-exec-in-namespaces
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: ["*"]
        kinds: ["*"]
    scope: Namespaced
  parameters:
    forbiddenNamespaces:
      - production
      - staging
```

Key data and checks for `CONNECT` validation:

- Use `input.review.operation == "CONNECT"` to check for `CONNECT` operations.
- User information is available in `input.review.userInfo.username` and `input.review.userInfo.groups`.
- The namespace is available in `input.review.namespace`.

Image signature verification

DP supports container image signature verification using Cosign.
Container image signature verification allows you to ensure their integrity (that the image has not been modified since its creation) and authenticity (that the image was created by a trusted source). You can enable container image signature verification in the cluster using the policies.verifyImageSignatures parameter of the SecurityPolicy.

Images are signed by creating a special tag in the container registry that contains the image signature. The signature is generated for the digest (hash) of your image.
If your image is `my-repo/app:latest` with the hash `sha256:abc123EXAMPLE`, the tag `my-repo/app:sha256-abc123EXAMPLE.sig` will appear in the image store.

Therefore, the image signing process consists of calculating and publishing an additional tag to the container registry, without modifying the existing image.
After signing the image, there is no need to push it to the image store again. You only need to log in to the container registry with write access.

To sign an image with Cosign, do the following:

1. Make sure that Cosign version 2 or lower is used:

   Check the version: `cosign version`.

   ```
   cosign version
   ```

2. Generate a key pair (public and private):

   ```
   cosign generate-key-pair
   ```

3. Sign the image in the container registry using the generated private key:

   ```
   cosign sign --key <KEY> <REGISTRY_IMAGE_PATH>
   ```

   Here:

- `<REGISTRY IMAGE PATH>` is the path to the image that needs to be specified at startup, for example, `registry.private.io/labs/application/image:latest`.

To enable container image signature verification in a DP cluster:

1. Use the `policies.verifyImageSignatures` parameter in SecurityPolicy and specify the generated public key.

   Example SecurityPolicy configuration for verifying signatures of container images in `registry.private.ru`, located under `/labs/application/`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: SecurityPolicy
   metadata:
     name: verify-image-test
   spec:
     enforcementAction: Deny
     match:
       namespaceSelector:
         labelSelector:
           matchLabels:
             example-security-policy/enabled: true
     policies:
       allowHostIPC: true
       allowHostNetwork: true
       allowHostPID: false
       allowPrivilegeEscalation: true
       allowPrivileged: false
       allowRbacWildcards: true
       verifyImageSignatures:
         - publicKeys:
             - |-
               -----BEGIN PUBLIC KEY-----
               ...
               -----END PUBLIC KEY-----
           reference: registry.private.io/labs/application/*
   ```

   The label name specified in `match.namespaceSelector.labelSelector.matchLabels` can be any name. The label in the policy selector must match the label on the corresponding namespace.

   For more information, see Configuring policy selectors.

2. Create an OperationPolicy that restricts running pods from third-party registries:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: OperationPolicy
   metadata:
     name: test-operation-policy
   spec:
     enforcementAction: Deny
     match:
       namespaceSelector:
         labelSelector:
           matchLabels:
             example-operation-policy/enabled: "true"
     policies:
       allowedRepos:
       - registry.private.io/labs/application/
   ```

   The label name specified in `match.namespaceSelector.labelSelector.matchLabels` can be any name. The label in the policy selector must match the label on the corresponding namespace.

   For more information, see Configuring policy selectors.

3. Add a label to the namespace where signature verification should be enabled (specify your namespace):

```
d8 k label ns <NAMESPACE> example-security-policy/enabled=true
```

4. Add a label to the namespace where running pods from third-party registries should be restricted (specify your namespace):

```
d8 k label ns <NAMESPACE> example-operation-policy/enabled=true
```

5. To verify how image signing works, deploy pods in the namespace with signed and unsigned images (specify your namespace):

```
d8 k -n <NAMESPACE> run signed-pod --image=<SIGNED_IMAGE>
d8 k -n <NAMESPACE> run unsigned-pod --image=<UNSIGNED_IMAGE>
```

According to this policy, if any container image address matches the `reference` parameter value and the image is unsigned, or the signature does not match the specified keys, pod creation will be denied.

Example error output when creating a pod with an image that fails signature verification:

```
[verify-image-signatures] Image signature verification failed: nginx:1.17.2
```

Using alternative security policy management tools

If you use an alternative solution for security policy management in a DP cluster (for example, Kyverno), configure exceptions for the following namespaces:

- `kube-system`
- all namespaces with the `d8-*` prefix (for example, `d8-system`)

Without these exceptions, policies may block or disrupt system component operations.

Additional resources

- Module `admission-policy-engine` documentation
- Module `deckhouse` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Container image vulnerability scanning

Deckhouse Platform (DP) provides a built-in tool for automatically detecting vulnerabilities in container images based on the Trivy project.

Vulnerability scanning

DP scans all container images used by cluster pods. The checks include:

- Known vulnerabilities (CVEs) in the images in use.
- Compliance with CIS benchmarks (compliance checks).

Scanning uses public vulnerability databases.

Scanning in namespaces

By default, only the `default` namespace is scanned.

To enable scanning for a specific namespace, add the label `security-scanning.deckhouse.io/enabled=""` to it.

As soon as at least one namespace with this label is detected in the cluster, scanning of the `default` namespace stops. To re-enable scanning for the `default` namespace, add the label with the following command:

```
d8 k label namespace default security-scanning.deckhouse.io/enabled=""
```

In the current version, there is no option to limit the list of resources to be scanned within a namespace. DP scans **all resources** in a namespace labeled with `security-scanning.deckhouse.io/enabled=""`.

Start conditions and scanning process

Scanning starts:

- automatically every 24 hours,
- when components with new images are deployed in namespaces with scanning enabled.

Scanning occurs according to the following process:

1. In the namespace of each scanned resource, a VulnerabilityReport object is created.
2. This object contains the annotation `trivy-operator.aquasecurity.github.io/report-ttl`, which specifies the report's time-to-live (default: `24h`).
3. When the TTL expires, the VulnerabilityReport object is deleted, and the scan is run again.

Forcing a rescan

To manually trigger a rescan of a resource, you can use either of the following methods:

- Update the `trivy-operator.aquasecurity.github.io/report-ttl` annotation with a short time-to-live value.

  Example:

  ```
  d8 k annotate VulnerabilityReport -n <NAMESPACE> <REPORT_NAME> trivy-operator.aquasecurity.github.io/report-ttl=1s --overwrite
  ```

- Delete the VulnerabilityReport object from the namespace containing the scanned resource.

  Example:

  ```
  d8 k delete VulnerabilityReport -n <NAMESPACE> <REPORT_NAME>
  ```

For details on how to view scan results, see Scanning container images for vulnerabilities.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Certificate management

Deckhouse Platform (DP) provides built-in tools for managing TLS certificates in the cluster and supports:

- Requesting certificates from all supported sources, such as Let's Encrypt, HashiCorp Vault, Venafi.
- Issuing self-signed certificates.
- Automatic renewal and expiration monitoring.
- Deploying `cm-acme-http-solver` on master nodes and dedicated nodes.

This page describes the available certificate management features in DP and the procedure for working with certificate issuers.

> Examples of certificate configuration, usage of the `tls-acme` annotation, and secure handling of credentials are described in Using TLS certificates.

Monitoring

DP exports metrics to Prometheus, allowing you to monitor:

- Certificate expiration dates
- Certificate renewal status

Access roles

DP provides several predefined roles for accessing resources:

Role	Permissions
`User`	View Certificate and Issuer resources in available namespaces, as well as global ClusterIssuer resources.
`Editor`	Manage Certificate resources in available namespaces.
`Admin`	Manage Issuer resources in available namespaces.
`ClusterEditor`	Manage Certificate and Issuer resources in all namespaces, as well as ClusterIssuer resources.
`SuperAdmin`	Manage internal system objects.

Working with certificate issuers

DP supports the following default certificate issuers (ClusterIssuer):

- `letsencrypt`: Issues TLS certificates using the public CA Let’s Encrypt and ACME HTTP validation. Suitable for automatically obtaining trusted certificates for most public services. Configuration details can be found in the official `cert-manager` documentation.
- `letsencrypt-staging`: Similar to `letsencrypt`, but uses Let’s Encrypt’s staging server. Useful for debugging configuration and testing the certificate issuance process. More details available in the Let’s Encrypt staging docs.
- `selfsigned`: Issues self-signed certificates. Useful when external trust is not required (for example, for internal services).
- `selfsigned-no-trust`: Also issues self-signed certificates but without automatically adding the root certificate to the trust store. Suitable for manual trust management.

In some cases, additional types of ClusterIssuer may be needed:

- If you want to use a Let’s Encrypt certificate with DNS validation via a third-party DNS provider.
- if you need to use a certificate authority (CA) other than Let’s Encrypt. See the full list of supported CAs in the `cert-manager` documentation.

Adding a ClusterIssuer with DNS-01 validation via webhook

To verify domain ownership with Let’s Encrypt using the `DNS-01` method, the `cert-manager` module must be able to create TXT records in the DNS zone associated with the domain.

`cert-manager` includes built-in support for popular DNS providers such as AWS Route53, Google Cloud DNS, Cloudflare, and others. A full list is available in the official `cert-manager` documentation.

The `cert-manager` module in Deckhouse Platform can automatically create ClusterIssuers for Cloudflare, Amazon Route53, DigitalOcean, Google Cloud DNS, and Yandex Cloud DNS when the corresponding module settings are filled in. For Yandex Cloud DNS, follow the steps in Issuing a DNS wildcard certificate using Yandex Cloud DNS.

If your provider is not directly supported, you can configure a webhook and deploy a custom ACME handler in the cluster that performs the necessary DNS record updates, then create a `ClusterIssuer` with `dns01.webhook` solver settings according to the webhook documentation.

Adding a ClusterIssuer that uses a custom certificate authority (CA)

1. Generate the certificate:

```
openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
```

2. Create a secret in the `d8-cert-manager` namespace with an arbitrary name, containing the certificate files.
   - Example of creating a secret using the `d8 k` command:

     ```
     d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --cert="rootCACert.pem"
     ```

   - Example of creating a secret from a YAML file (the certificate file contents must be Base64-encoded):

     ```
     apiVersion: v1
     data:
       tls.crt: <output of `cat rootCACert.pem | base64 -w0`>
       tls.key: <output of `cat rootCAKey.pem | base64 -w0`>
     kind: Secret
     metadata:
       name: internal-ca-key-pair
       namespace: d8-cert-manager
     type: Opaque
     ```

3. Create a ClusterIssuer with any name using the created secret:

   ```
   apiVersion: cert-manager.io/v1
   kind: ClusterIssuer
   metadata:
     name: inter-ca
   spec:
     ca:
       secretName: internal-ca-key-pair   # Name of the created secret.
   ```

You can now use the created ClusterIssuer to issue certificates for all DP components or for a specific component.

For example, to use this ClusterIssuer for issuing certificates for all DP components, set its name in the global parameter `clusterIssuerName`:

```
spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
        mode: CertManager
      publicDomainTemplate: '%s.<public_domain_template>'
  version: 1
```

Adding an Issuer and ClusterIssuer using HashiCorp Vault to request certificates

To configure certificate issuance using Vault, refer to the HashiCorp documentation.

After configuring the PKI and enabling Kubernetes authentication, follow these steps:

1. Create a ServiceAccount and copy the reference to its secret:

   ```
   d8 k create serviceaccount issuer

   ISSUER_SECRET_REF=$(d8 k get serviceaccount issuer -o json | jq -r ".secrets[].name")
   ```

2. Create the Issuer resource:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    server: http://vault.default.svc.cluster.local:8200
    # Defined during PKI configuration.
    path: pki/sign/example-dot-com
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: $ISSUER_SECRET_REF
          key: token
EOF
```

3. Create the Certificate resource to obtain a TLS certificate signed by Vault CA:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # Domains must match those configured in the PKI in Vault.
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

Additional resources

- Global parameters
- Module `cert-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integration with software and security solutions

Deckhouse Platform supports integration with Kaspersky Unified Monitoring and Analysis Platform (KUMA), a unified monitoring and analysis system by Kaspersky Lab. As part of the integration, security events and audit logs from the cluster are sent to KUMA for further analysis.

Sending logs to KUMA

To send logs to KUMA, configure log collection and delivery in Deckhouse Platform using the following resources:

- ClusterLogDestination: Defines log storage parameters.

- ClusterLoggingConfig: Defines cluster log collection parameters.

> **i** On the KUMA side, configure the appropriate resources for receiving events.

Example ClusterLogDestination and ClusterLoggingConfig configurations

In the following examples, you can find ClusterLogDestination and ClusterLoggingConfig configuration options for sending logs in various formats.

- In JSON format over UDP
- In JSON format over TCP
- In CEF format over TCP
- In Syslog format over TCP
- Apache Kafka

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-udp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: UDP
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-udp-json
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-tcp-json
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-cef
spec:
  type: Socket
  socket:
    extraLabels:
      cef.name: d8
      cef.severity: "1"
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "CEF"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-cef
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-syslog
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "Syslog"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-syslog
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-kafka
spec:
  type: Kafka
  kafka:
    bootstrapServers:
      - kafka-address:9092 # Replace as needed.
    topic: k8s-logs
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  destinationRefs:
    - kuma-kafka
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values:
        - kubelet.*
  type: File
```

KCS configuration features

Kaspersky Container Security (KCS) requires extended access rights to the OS kernel in order to monitor the execution environment. Specifically, the `kcs-ih` (Image Hub) component must run in privileged mode. To do this, certain process privilege management mechanisms at the Linux OS kernel level must be disabled. For more details, see the Kaspersky Container Security documentation.

To enable KCS runtime monitoring in a Deckhouse Platform cluster, assign the label `security.deckhouse.io/pod-policy=privileged` to the namespace where KCS components are located.

Example command for assigning a label:

```
d8 k label namespace kcs security.deckhouse.io/pod-policy=privileged
```

Antivirus scanning exclusions for nodes

If antivirus software is installed on Deckhouse Platform cluster nodes (for example, Kaspersky Endpoint Security for Linux, KESL), you may need to exclude Deckhouse service directories from scanning to avoid false positives.

List of Deckhouse service directories (also available CSV format):

Directory	Purpose
`/etc/cni/` (any node)	CNI module configuration files
`/etc/kubernetes` (any node)	Static pod manifests, PKI certificate files
`/mnt/kubernetes-data` (master node)	Only present in clusters deployed in the cloud when a separate disk is used for the etcd database
`/mnt/vector-data` (any node, `log-shipper` module)	Service data for log delivery status
`/opt/cni/bin/` (any node)	CNI module executables
`/opt/deckhouse/bin/` (any node)	Executables required for Deckhouse Platform operation
`/var/lib/bashible` (any node)	Node configuration files
`/var/lib/containerd` (any node)	Stores CRI-related data (for example, containerd). Contains container image layers, filesystem snapshots, metadata, logs, and other container information
`/var/lib/deckhouse/` (master node)	Deckhouse Platform module files dynamically loaded from the image registry
`/var/lib/etcd` (master node)	etcd database
`/var/lib/kubelet/` (any node)	kubelet configuration files
`/var/lib/upmeter` (master node, `upmeter` module)	`upmeter` module database
`/var/log/containers` (any node)	Container logs (when using containerd)
`/var/log/pods/` (any node)	Logs of all pod containers running on the node

KESL configuration recommendations

To ensure Deckhouse Platform functions correctly with KESL installed, follow these steps:

1. Disable the following KESL tasks:
   - `Firewall Management (ID: 12)`
   - `Web Threat Protection (ID: 14)`
   - `Network Threat Protection (ID: 17)`
   - `Web Control (ID: 26)`
   - `Mail Monitoring ( ID: 35 )`

> **i** The list of tasks may differ in future KESL versions.

2. Make sure node resources meet the requirements of:
    - Deckhouse Platform
    - KESL
3. For performance optimization, follow the official Kaspersky recommendations.

Additional resources

- Module `log-shipper` custom resources
- Module `upmeter` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Automatic distribution of secrets across namespaces

Deckhouse Platform (DP) lets you automatically distribute secrets across all namespaces in the cluster. This helps eliminate the need to manually copy secrets in CI/CD pipelines — for example, when pulling container images or working with Ceph RBD storage.

How it works

- Deckhouse monitors secrets in the `default` namespace that have the special label `secret-copier.deckhouse.io/enabled: ""`.
- These secrets are automatically copied to all namespaces in the cluster.
- When the original secret in the `default` namespace is updated, the new content is propagated to all namespaces.
- If the original secret in `default` is deleted, all of its copies in other namespaces are also deleted.
- If a secret copy is manually modified in a namespace, it will be automatically overwritten with the content of the original secret from `default`.
- When new namespaces are created, all secrets from `default` labeled with `secret-copier.deckhouse.io/enabled: ""` are automatically copied into them.
- There's a regular nightly synchronization that ensures all secrets remain up to date.
- Deckhouse Platform never creates, updates or deletes the secret `d8-system/cluster-key` this way: it holds the cluster key of Deckhouse Platform licensing.
- Secrets of the `bootstrap.kubernetes.io/token` type are not copied. The copies of such secrets that earlier Deckhouse Platform versions made are deleted.
- Deckhouse Platform updates and deletes only the copies it made. If a namespace already has a secret with the same name that Deckhouse Platform did not create, the secret is left as is and is not copied into that namespace.

Incompatibility with multitenancy mode

The automatic secret distribution mechanism is not compatible with the multitenancy mode.

Multitenancy mode creates isolated environments for users within projects. Automatically distributing secrets to all namespaces may violate this security model. If sensitive data accidentally reaches a user's isolated environment, it may lead to data exposure.

If you need to provide a shared certificate (for example, a wildcard certificate for an internal environment) or a shared registry access token, do not use automatic distribution. Instead, include such secrets in the project template. A cluster administrator must define them in the project configuration.

Configuring automatic secret distribution

1. Create a Secret resource in the `default` namespace.
2. Add the label `secret-copier.deckhouse.io/enabled: ""`.
3. Apply the resource using the `d8 k apply` command. It will then be automatically distributed to all namespaces.

⚠ Only secrets from the `default` namespace are automatically distributed. Secrets with the label `secret-copier.deckhouse.io/enabled: ""` created in any other namespace will be automatically deleted.

Distributing secrets to selected namespaces

To copy a secret to specific namespaces only, use the annotation with a label selector `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. In this case, the secret will be copied only to namespaces that match the specified selector.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Monitoring in Deckhouse Platform

Deckhouse Platform (DP) provides a Kubernetes monitoring solution based on **Prometheus** and **Grafana**. DP automatically configures metrics collection in the cluster from nodes, pods, and key cluster components (etcd, kube-apiserver, CoreDNS), which enables preset dashboards for analyzing CPU, memory, disk, and network usage.

Cluster monitoring is enabled by default in the `Default` and `Managed` module bundles.

All components, including Prometheus and Alertmanager, operate in a fault-tolerant mode and can be used in cloud environments and on bare-metal servers.

The principles of Prometheus operation is covered in Configuring a system for collecting and storing metrics.

Several types of monitoring are implemented in DP:

- Hardware resource monitoring
- Kubernetes monitoring
- Ingress monitoring
- Control plane monitoring
- Network interaction monitoring
- Extended monitoring
- Cluster SLA monitoring

DP includes an alerting system that supports sending event notifications, including to external systems.

Hardware resource monitoring

Tracking of cluster hardware resource capacity is provided with graphs showing utilization of:

- CPU
- Memory
- Disk
- Network

Graphs are available with aggregation by:

- Pods
- Controllers
- Namespaces
- Nodes

Kubernetes monitoring

The module `monitoring-kubernetes` is designed for basic cluster node monitoring.

It provides secure metrics collection and offers a basic set of rules for monitoring:

- Current container runtime version (docker, containerd) on the node and its compliance with versions allowed for use.
- Overall cluster monitoring subsystem health (Dead man’s switch).

- Available file descriptors, sockets, free space, and inodes.
- Operation of `kube-state-metrics`, `node-exporter`, `kube-dns`.
- Cluster node state (NotReady, drain, cordon).
- Time synchronization state on nodes.
- Cases of prolonged CPU steal exceeding.
- Conntrack table state on nodes.
- Pods with incorrect state (as a possible consequence of kubelet issues) and more.

Ingress monitoring

Statistics collection for `ingress-nginx` in Prometheus is implemented with detailed metrics (response time, codes, geography, etc.), available in different dimensions (namespace, vhost, ingress). Data is visualized in Grafana with interactive dashboards. Detailed description is available in the section about Ingress monitoring.

The module is enabled by default in the `Default` and `Managed` module bundles.

Disabling collection of detailed statistics from Ingress resources

By default, DP collects detailed statistics from all Ingress resources in the cluster, which generates a high load on the monitoring system.

To disable statistics collection, add the label `ingress.deckhouse.io/discard-metrics: "true"` to the corresponding namespace or Ingress resource.

- Example of disabling statistics (metrics) collection for all Ingress resources in the `review-1` namespace:

  ```
  d8 k label ns review-1 ingress.deckhouse.io/discard-metrics=true
  ```

- Example of disabling statistics (metrics) collection for all `test-site` Ingress resources in the `development` namespace:

  ```
  d8 k label ingress test-site -n development ingress.deckhouse.io/discard-metrics=true
  ```

Control plane monitoring

Control plane monitoring is provided by the `control-plane-manager` module. It organizes secure metrics collection and provides a basic set of monitoring rules for the following cluster components:

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kube-etcd

Cluster monitoring

DP securely collects monitoring metrics and configures rules.

DP monitoring capabilities:

- Monitoring current container runtime version (containerd) on the node and its compliance with versions allowed for use in DP.
- Monitoring cluster monitoring subsystem health (“Dead man’s switch”).
- Monitoring available file descriptors, sockets, free space, and inodes.
- Monitoring cluster node state (NotReady, drain, cordon).
- Operation of `kube-state-metrics`, `node-exporter`, `kube-dns`.
- Monitoring time synchronization state on nodes.
- Monitoring cases of prolonged CPU steal exceeding.
- Monitoring Conntrack table state on nodes.
- Monitoring pods with incorrect state (as a possible consequence of kubelet issues).
- Monitoring control plane components (implemented by the `control-plane-manager` module).

Extended monitoring mode

DP supports an extended monitoring mode via the `extended-monitoring` module, allowing you to configure:

- Monitoring secrets in the cluster (Secret objects) and TLS certificate expiration in them.
- Collecting Kubernetes cluster events as metrics.

- Monitoring container image availability in registry used by controllers (Deployments, StatefulSets, DaemonSets, CronJobs).
- Monitoring objects in namespaces that have the `extended-monitoring.deckhouse.io/enabled=""` label.

The module can send alerts based on the following metrics:

- Free space and inodes on node disks
- Node utilization
- Pod and container image availability
- Certificate expiration
- Other cluster events

Alerts

Monitoring in DP includes event notifications. The standard delivery includes a set of basic warnings covering cluster state and its components. There is also the ability to add custom alerts.

For the list of all available alerts in the DP monitoring system, refer to the corresponding documentation page.

Sending alerts to external systems

DP supports sending alerts using Alertmanager:

- Via SMTP protocol
- To PagerDuty
- To Slack
- To Telegram
- Via Webhook
- Through any other channels supported in Alertmanager

Examples of DP monitoring integration with external systems are available in Configuring integrations.

Cluster SLA monitoring

Availability assessment in DP is performed by the `upmeter` module.

Composition of the `upmeter` module:

- **agent**: Runs on master nodes and performs availability probes, sends results to the server.
- **upmeter**: Collects results and maintains an API server for their retrieval.
- **front**:
  - **status**: Shows availability level for the last 10 minutes (requires authorization, but it can be disabled).
  - **webui**: Shows a dashboard with statistics on probes and availability groups (requires authorization).
- **smoke-mini**: Maintains continuous *smoke testing* using StatefulSet.

The module sends about 100 metric readings every 5 minutes. This value depends on the number of enabled Deckhouse Platform modules.

Additional resources

- The list of alerts
- Integration with external monitoring systems
- Web interface of the monitoring
- Web interface of the platform
- Module `prometheus` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring network interaction and cluster node monitoring

Network interaction monitoring

DP can perform monitoring of network interaction between all cluster nodes, as well as between cluster nodes and external hosts. When monitoring is configured, each node sends ICMP packets twice per second to all other cluster nodes (and to optional external nodes) and exports data to the monitoring system.

Features of network interaction monitoring:

- Automatically checks the availability of all cluster nodes (and, optionally, external systems) using ICMP (ping) — testing is started every two seconds.
- All results are exported in metrics format to the Prometheus monitoring system.
- Included is a ready-made dashboard for Grafana, where current availability, delay schedules, and potential network connectivity issues are visualized in real time (for details, refer to Grafana).
- Allows you to quickly identify nodes with degraded connectivity and speeds up the response to incidents.

The `monitoring-ping` module tracks any changes to the `.status.addresses` field of a node. If changes are detected, a hook is triggered that collects the complete list of node names and their addresses, and passes it to the DaemonSet, which recreates the pods. Thus, `ping` checks always the current list of nodes.

⚠ The `monitoring-ping` module must be enabled.

Adding additional IP addresses for monitoring

To add additional monitoring IP addresses, use the `externalTargets` parameter of the `monitoring-ping` module.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings:
    externalTargets:
    - name: google-primary
      host: 8.8.8.8
    - name: yaru
      host: ya.ru
    - host: youtube.com
```

i The `name` field is used in Grafana to display related data. If the `name` field is not specified, the required `host` field is used.

Cluster node monitoring

To enable cluster node monitoring, you need to enable the `monitoring-kubernetes` module if it’s not already enabled. You can enable cluster monitoring in the Deckhouse web interface, or using the following command:

```
d8 system module enable monitoring-kubernetes
```

In a similar way, you can enable the `extended-monitoring` module.

Additional resources

- Module `console` documentation
- Module `extended-monitoring` documentation

- Module `monitoring-kubernetes` documentation
- Module `monitoring-ping` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Writing Prometheus data to longterm storage

Prometheus supports the remote_write mechanism for sending data from a local Prometheus instance to a separate long-term storage (for example, VictoriaMetrics). In DP, support for this mechanism is implemented using the PrometheusRemoteWrite custom resource.

> For VictoriaMetrics, detailed information about ways to send data to vmagent can be found in the VictoriaMetrics documentation.

Example of minimal PrometheusRemoteWrite

Below is an example of minimal configuration for the PrometheusRemoteWrite custom resource:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
```

Example of advanced PrometheusRemoteWrite

Below is an example of advanced configuration for the PrometheusRemoteWrite custom resource:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
  basicAuth:
    username: username
    password: password
  writeRelabelConfigs:
  - sourceLabels: [__name__]
    action: keep
    regex: prometheus_build_.*|my_cool_app_metrics_.*
  - sourceLabels: [__name__]
    action: drop
    regex: my_cool_app_metrics_with_sensitive_data
```

For a complete description of the custom resource fields, refer to the prometheus module documentation.

Additional resources

- Module `prometheus` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

External access configuration

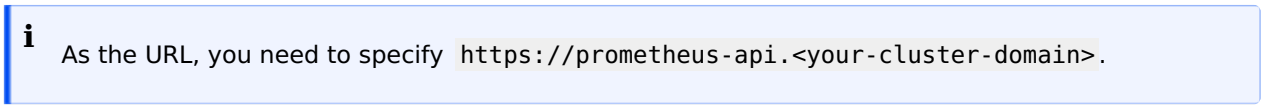
Connecting Prometheus to Third-Party Grafana

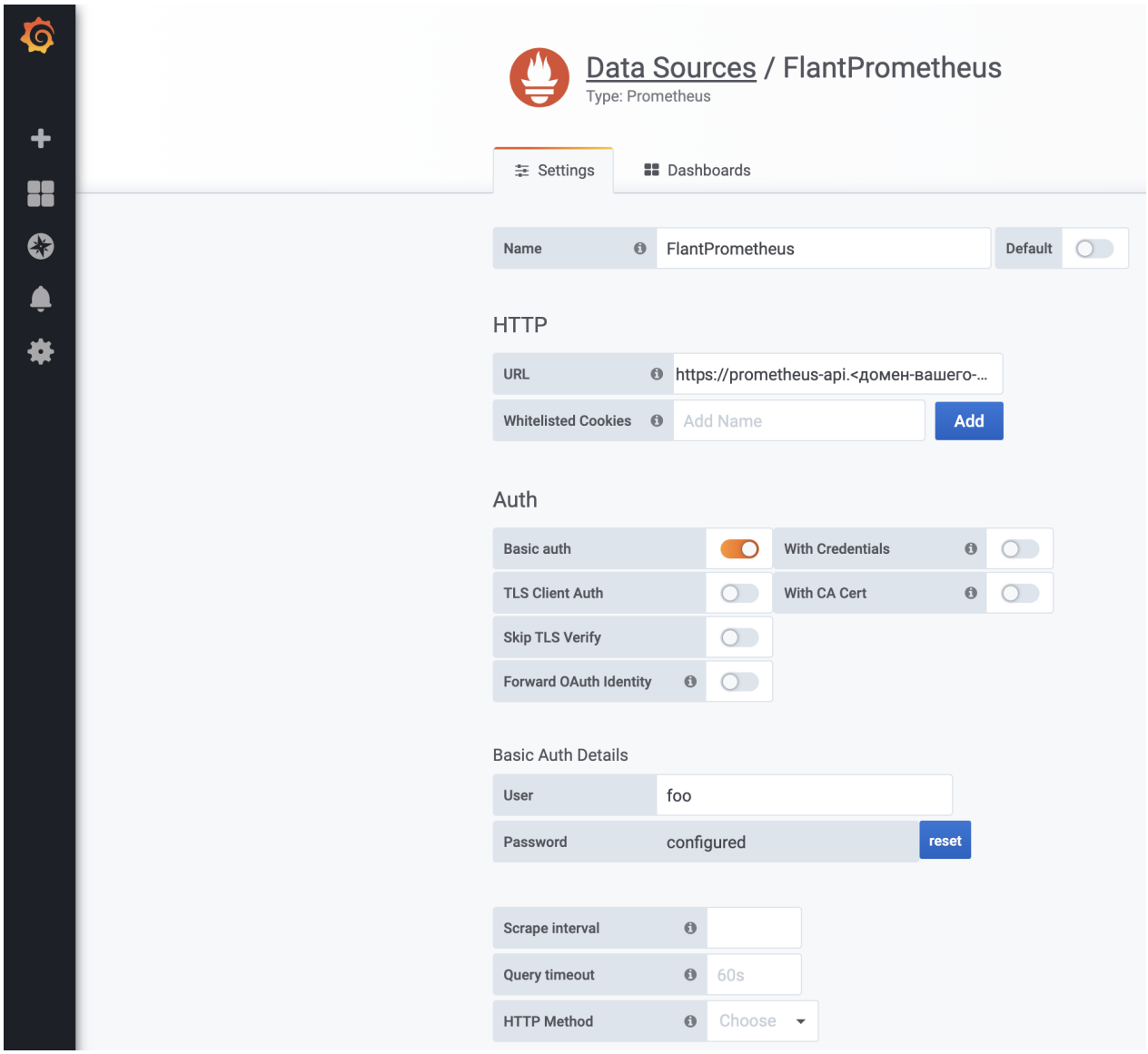
Each `ingress-nginx-controller` has certificates that, when specified as client certificates, will allow connection to Prometheus. All you need to do is create an additional `Ingress` resource.

> The example below assumes that the Secret `example-com-tls` already exists in the d8-monitoring namespace.
>
> The names for Ingress `my-prometheus-api` and Secret `my-basic-auth-secret` are provided as examples. Replace them with more suitable ones for your setup.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-prometheus-api
  namespace: d8-monitoring
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: my-basic-auth-secret
    nginx.ingress.kubernetes.io/app-root: /graph
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_ssl_certificate /etc/nginx/ssl/client.crt;
      proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
      proxy_ssl_protocols TLSv1.2;
      proxy_ssl_session_reuse on;
spec:
  ingressClassName: nginx
  rules:
  - host: prometheus-api.example.com
    http:
      paths:
      - backend:
          service:
            name: prometheus
            port:
              name: https
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - prometheus-api.example.com
    secretName: example-com-tls
---
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-auth-secret
  namespace: d8-monitoring
type: Opaque
data:
  # Basic-auth string is hashed using htpasswd.
  auth: Zm9vOiRhcHIxJE9GRzNYeWJwJGNrTDBGSERBa29YWUlsSDkuY3lzVDAK  # foo:bar
```

Add the data source to Grafana:





- **Basic authorization** is not a reliable security measure. It is recommended to implement additional security measures, such as specifying the `nginx.ingress.kubernetes.io/whitelist-source-range` annotation.
- Due to the need to create an Ingress resource in the system namespace, this connection method is **not recommended**. DP **does not guarantee** the preservation of this connection scheme's functionality due to its active continuous updates.
- This Ingress resource can be used to access the Prometheus API not only for Grafana but also for other integrations, such as Prometheus federation.

Connecting Third-Party Applications to Prometheus

The connection to Prometheus is secured using kube-rbac-proxy. To connect, create a `ServiceAccount` with the necessary permissions.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app:prometheus-access
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["prometheuses/http"]
  resourceNames: ["main", "longterm"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app:prometheus-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app:prometheus-access
subjects:
- kind: ServiceAccount
  name: app
  namespace: default
```

Execute the request using the `curl` command:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app-curl
  namespace: default
spec:
  template:
    metadata:
      name: app-curl
    spec:
      serviceAccountName: app
      containers:
      - name: app-curl
        image: curlimages/curl:7.69.1
        command: ["sh", "-c"]
        args:
        - >-
          curl -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" -k -f
          https://prometheus.d8-monitoring:9090/api/v1/query_range?
query=up\&start=1584001500\&end=1584023100\&step=30
      restartPolicy: Never
  backoffLimit: 4
```

The `Job` should complete successfully.

Metrics Collection via Gateway (Pushgateway)

Prometheus, which is the foundation of the DP monitoring system, primarily uses a pull model for

metrics collection. In this approach, DP polls metric exporters. When applying the pull model is difficult, for example, for services without a permanent network interface, you can use metrics collection via a gateway (Pushgateway). Pushgateway allows such tasks to send metrics themselves, which can then be collected by Prometheus. It’s important to note that Pushgateway can become a single point of failure and bottleneck in the system. How to send metrics from an application to Pushgateway can be learned from the Prometheus documentation.

Example of configuring metrics collection via gateway (Pushgateway):

- Enable and configure the `prometheus-pushgateway` module.

  You can enable the module in the web interface (Deckhouse Console), or using the following command:

  ```
  d8 system module enable prometheus-pushgateway
  ```

- Specify the gateway names in the `instances` parameter of the `prometheus-pushgateway` module through the web interface, or using the following command:

  ```
  d8 k edit mc prometheus-pushgateway
  ```

  Example module configuration:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: prometheus-pushgateway
  spec:
    version: 1
    enabled: true
    settings:
      instances:
      - first
      - second
      - another
  ```

  The address of the PushGateway instance named `first` from a pod container will be: `http://first.kube-prometheus-pushgateway:9091`.

- Check metrics sending.

  Example of sending metrics via curl:

  ```
  echo "test_metric{env="dev"} 3.14" | curl --data-binary @- http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp
  ```

- Check that the metric appeared in the monitoring system. It will be available 30 seconds after data collection.

  Example PromQL query:

  ```
  test_metric{container="prometheus-pushgateway", env="dev", exported_job="myapp",
      instance="10.244.1.155:9091", job="prometheus-pushgateway", pushgateway="prometheus-pushgateway", tier="cluster"} 3.14
  ```

Removing Metrics from Gateway (Pushgateway)

Example of removing all metrics from group `{instance="10.244.1.155:9091",job="myapp"}` via curl:

```
curl -X DELETE http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp/instance/10.244.1.155:9091
```

Note that PushGateway stores received metrics in memory. When the PushGateway pod restarts, all metrics will be lost.

Additional resources

- Module `prometheus-pushgateway` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring a system for collecting and storing metrics

> Starting from version 1.71, Deckhouse Platform uses Deckhouse Prom++ instead of Prometheus.

Prometheus capabilities

Prometheus collects metrics and executes rules:

- For each *target* (monitoring target) at a specified interval `scrape_interval`, Prometheus makes an HTTP request to this *target*, receives metrics in its own format in response, and stores them in its database.
- Every `evaluation_interval` it processes rules (*rules*), based on which:
  - it sends alerts;
  - or stores new metrics (result of rule execution) in its database.

Prometheus operation

Prometheus is installed by the `prometheus` module of DP, which performs the following functions:

- Defines the following custom resources:
  - `Prometheus`: Defines the *Prometheus* installation (cluster).
  - `ServiceMonitor`: Defines how to collect metrics from services.
  - `Alertmanager`: Defines the *Alertmanager* cluster.
  - `PrometheusRule`: Defines the list of *Prometheus rules*.
- Monitors these resources, and also:
  - Generates `StatefulSet` with *Prometheus* itself.
  - Creates secrets with configuration files necessary for Prometheus operation (`prometheus.yaml` — Prometheus configuration, and `configmaps.json` — configuration for `prometheus-config-reloader`).
  - Monitors `ServiceMonitor` and `PrometheusRule` resources and updates *Prometheus* configuration files by modifying secrets based on them.

The module can be enabled using the following ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings:
    auth:
      password: xxxxxx
    retentionDays: 7
    storageClass: rbd
    nodeSelector:
      node-role/monitoring: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: monitoring
```

Using the `global.modules.storageClass` parameter for Prometheus

This module uses the global `global.modules.storageClass` parameter as the default StorageClass when creating a new PersistentVolumeClaim (if the `storageClass` parameter is not specified in the module's configuration).

If a Prometheus PVC already exists, changing the global parameter does not affect the existing PVC or trigger disk recreation. The module continues to use the existing PVC and its StorageClass.

If the module configuration defines its own `storageClass` or `longtermStorageClass` value, the module-specific value takes precedence over the global default.

⚠ Changing the `storageClass` or `longtermStorageClass` parameter in the module configuration deletes and recreates the existing PVC. All data will be lost. Back up your data before making this change.

A complete description of all settings is available in the prometheus module documentation.

Additional resources

- Global parameters
- Module `prometheus` documentation

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster SLA monitoring

DP can collect statistics about the availability of cluster components and DP components themselves. This data allows evaluating SLA compliance and provides availability information in the web interface.

Additionally, using the UpmeterRemoteWrite custom resource, you can export availability metrics via the Prometheus Remote Write protocol.

To start collecting availability metrics and activate the interface, enable the `upmeter` module in the Deckhouse web interface or use the following command:

```
d8 system module enable upmeter
```

Module configuration

The `upmeter` module is configured using the `upmeter` ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  version: 3
  enabled: true
  settings:
```

A complete list of all settings is available in the module documentation.

Interface

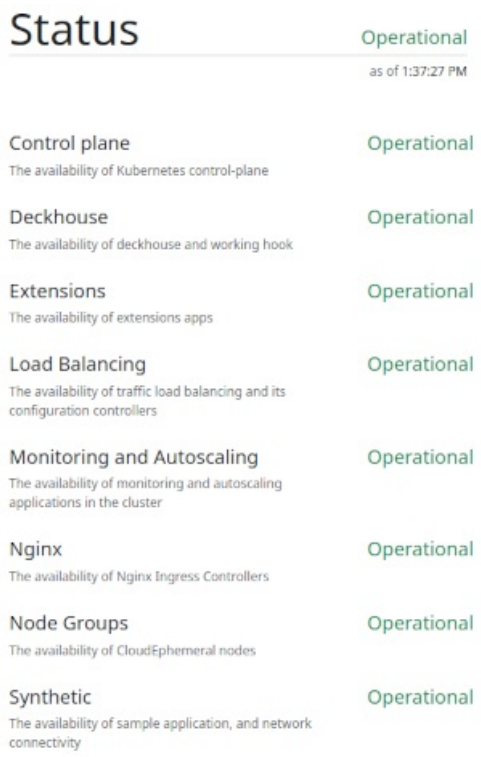
DP provides two web interfaces for availability assessment:

- Status page.

You can get the page address in the web interface on the main page in the “Tools” section (Status page block), or by running the command:

```
d8 k -n d8-upmeter get ing status -o jsonpath='{.spec.rules[*].host}'
```

Example of the status page web interface:



- Component availability page.

  You can get the page address in the web interface on the main page in the “Tools” section (Component availability block), or by running the command:

  ```
  d8 k -n d8-upmeter get ing webui -o jsonpath='{.spec.rules[*].host}'
  ```

  Example of the component availability page:



Status metrics export

Example of UpmeterRemoteWrite configuration for exporting status metrics via the Prometheus Remote Write protocol:

```
apiVersion: deckhouse.io/v1
kind: UpmeterRemoteWrite
metadata:
  labels:
    heritage: upmeter
    module: upmeter
  name: victoriametrics
spec:
  additionalLabels:
    cluster: cluster-name
    some: fun
  config:
    url: https://upmeter-victoriametrics.whatever/api/v1/write
    basicAuth:
      password: "Cdp#Cd.OxfZsx4*89SZ"
      username: upmeter
  intervalSeconds: 300
```

Authentication

By default, the `user-authn` module is used for authentication. You can also configure authentication via `externalAuthentication`. If these options are disabled, the module will enable basic authentication with a generated password.

You can view the generated password using the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.webui.password'
```

To generate a new password, you need to delete the Secret:

```
d8 k -n d8-upmeter delete secret/basic-auth-webui
```

You can view the generated password for the status page using the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.status.password'
```

To generate a new password for the status page, you need to delete the secret:

```
d8 k -n d8-upmeter delete secret/basic-auth-status
```

> **Attention!** The `auth.status.password` and `auth.webui.password` parameters are no longer supported.

Behavior of upmeter pods

Upmeter tests create temporary pods to check that Kubernetes components are working. As a result, some pods are periodically deleted, remain in the `Pending` state, or move between nodes.

The following objects take part in the checks:

- `upmeter-probe-scheduler`: Checks the scheduler. The test creates a pod, schedules it to a node, and then deletes it.
- `upmeter-probe-controller-manager`: Checks `kube-controller-manager`. The test creates a StatefulSet and verifies that the StatefulSet created a pod. This test does not check pod placement on a node, so it creates a pod that cannot be scheduled and stays in the `Pending` state. Then the StatefulSet is deleted, and the test verifies that the spawned pod is deleted as well.
- `smoke-mini`: Checks network connectivity between nodes. Five StatefulSets with one replica each are created. The test checks connectivity between `smoke-mini` pods and `upmeter-agent` pods on master nodes. Once a minute, one of the `smoke-mini` pods is moved to another node.

Additional resources

- Module `console` documentation
- Module `upmeter` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring integrations

Redirecting alerts to Zabbix

Deckhouse Platform supports integration with the Zabbix monitoring system. For this purpose, an external script is used that receives alerts from Deckhouse via `kubectl` and sends them to Zabbix using the Zabbix agent.

The script requires:

- installed and configured zabbix-agent;
- utilities `bash`, `jq`;
- `kubectl` with a working configuration file and permissions to execute the `kubectl get clusteralerts` command.

Installation

1. Import the template into Zabbix:
   - In the Zabbix web interface, go to the “Data collection → Templates” section
   - Click “Import” and upload the `zbx_export_templates.yaml` file:

```
zabbix_export:
  version: '6.4'
  template_groups:
    - uuid: 7df96b18c230490a9a0a9e2307226338
      name: Templates
  templates:
    - uuid: 91d9cf9d023749a3997bc6429ef45fe1
      template: 'Module d8alerts'
      name: 'Deckhouse Alerts'
      groups:
        - name: Templates
      discovery_rules:
        - uuid: e9be18e9305e48e887c43e1ad9ea9c31
          name: 'Deckhouse Alert Discovery'
          key: d8alerts.discovery
          lifetime: 3d
          item_prototypes:
            - uuid: c4152d379d7a46cab23f7437cdb632c1
              name: 'Deckhouse alert: {#ALERTNAME} severity'
              key: 'd8alerts.severity[{#ALERTID}]'
              history: 1d
              trends: '0'
              description: |
                Summary: {#SUMMARY}

                {#DESCRIPTION}

                Labels: {#LABELS}
                To get more information: `kubectl describe clusteralert
{#ALERTID}`
```

```
            preprocessing:
              - type: REGEX
                parameters:
                  - (\d+)
                  - \1
            tags:
              - tag: Application
                value: deckhouse
            trigger_prototypes:
              - uuid: b27f8d03dc4c4d33bab4193a6ce57ae7
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 2 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) > 0'
                name: '{#ALERTNAME} is firing'
                priority: DISASTER
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
{#ALERTID}`
              - uuid: d1935044cf3f49df8bc53f738e36b683
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 3 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 4'
                name: '{#ALERTNAME} is firing'
                priority: HIGH
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
{#ALERTID}`
              - uuid: 0b134c599d9d4de79f906fbd8e749ec2
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 5 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 6'
                name: '{#ALERTNAME} is firing'
                priority: AVERAGE
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
{#ALERTID}`
              - uuid: 6967dc80d6414239b2d597447815048a
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 7 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 8'
                name: '{#ALERTNAME} is firing'
                priority: WARNING
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
```

```
{#ALERTID}`
             - uuid: 8e531a2e6a0f47849c1dffea9a1733e1
               expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 9 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 10'
               name: '{#ALERTNAME} is firing'
               priority: INFO
               description: |
                 Summary: {#SUMMARY}

                 {#DESCRIPTION}

                 Labels: {#LABELS}
                 To get more information: `kubectl describe clusteralert
{#ALERTID}`
```

- After installing the template, assign it to the group corresponding to the agents from which metrics will be collected.

2. Configure the Zabbix agent:
   - Copy the `d8alerts.conf` file to the directory specified in the `Include` parameter of the main Zabbix agent config (usually located at `/etc/zabbix/zabbix_agentd.d/`):

     ```
     # LLD of deckhouse cluster alerts
     UserParameter=d8alerts.discovery,/etc/zabbix/scripts/clusteralerts.sh
     discovery

     # Severity of a specific alert by its ID
     UserParameter=d8alerts.severity[*],/etc/zabbix/scripts/clusteralerts.sh
     severity "$1"
     ```

   - Copy the `clusteralerts.sh` script to the `/etc/zabbix/scripts/` directory:

```
#!/bin/bash

MODE="$1"
ALERT_ID="$2"

get_alerts_json() {
  kubectl get clusteralerts -o json
}

get_alert_by_id() {
  local id="$1"
  kubectl get clusteralerts "$id" -o json
}

if [[ "$MODE" == "discovery" ]]; then
  get_alerts_json | jq -c '{
      data: [.items[] | {
        "{#ALERTID}": .metadata.name,
        "{#ALERTNAME}": .alert.name,
        "{#SEVERITY}": (.alert.severityLevel | tonumber),
        "{#DESCRIPTION}": .alert.description,
        "{#SUMMARY}": .alert.summary,
        "{#LABELS}": (.alert.labels | to_entries | map("\(.key)=\(.value)") |
join(","))
      }]
    }'

elif [[ "$MODE" == "severity" && -n "$ALERT_ID" ]]; then
  get_alert_by_id "$ALERT_ID" | jq -r '.alert.severityLevel'

else
  echo "Invalid usage"
  exit 1
fi
```

- Make sure the script has execute permissions:

```
chmod +x /etc/zabbix/scripts/clusteralerts.sh
```

3. Check access:
   - Ensure the script has access to the cluster and can retrieve alert information:

     ```
     /etc/zabbix/scripts/clusteralerts.sh discovery
     ```

     The output should contain a list of alerts with their statuses and severity levels.
4. Restart the Zabbix agent to apply changes:

   ```
   systemctl restart zabbix-agent
   ```

Troubleshooting

1. Check Zabbix agent logs:

   ```
   tail -f /var/log/zabbix/zabbix_agentd.log
   ```

2. Check script operation:

   ```
   /etc/zabbix/scripts/clusteralerts.sh discovery
   /etc/zabbix/scripts/clusteralerts.sh severity "ALERT_ID"
   ```

Make sure the script executes correctly and returns expected data.

3. Check the `zabbix` user permissions. Ensure the agent runs the script as a user with the necessary cluster access rights:
   - The script runs as the `zabbix` user:

     ```
     sudo -u zabbix /etc/zabbix/scripts/clusteralerts.sh
     ```

   - The `KUBECONFIG` variable is available to the user:

     If the Kubernetes configuration file is not available by default, specify it explicitly. To do this, save the kubeconfig, for example, in `/etc/zabbix/kubeconfig` and add to the agent configuration:

     ```
     UserParameter=d8alerts.discovery,export
     KUBECONFIG=/etc/zabbix/kubeconfig;/etc/zabbix/scripts/clusteralerts.sh discovery
     ```

Sending alerts to Telegram

Alertmanager supports direct sending of alerts to Telegram.

Create a Secret in the `d8-monitoring` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: telegram-bot-secret
  namespace: d8-monitoring
stringData:
  token: "562696849:AAExcuJ8H6z4pTlPuocbrXXXXXXXXXXx"
```

Deploy the CustomAlertManager custom resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: telegram
spec:
  type: Internal
  internal:
    receivers:
      - name: telegram
        telegramConfigs:
          - botToken:
              name: telegram-bot-secret
              key: token
            chatID: -30490XXXXX
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: telegram
      repeatInterval: 12h
```

The `token` field in the Secret and `chatID` in the `CustomAlertmanager` resource need to be set to your own values. More details about the Telegram API.

Example of sending alerts to Slack with filter

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: slack
spec:
  internal:
    receivers:
    - name: devnull
    - name: slack
      slackConfigs:
      - apiURL:
          key: apiURL
          name: slack-apiurl
        channel: {{ dig .Values.werf.env .Values.slack.channel._default
.Values.slack.channel }}
        fields:
        - short: true
          title: Severity
          value: '{{`{{  .CommonLabels.severity_level }}`}}'
        - short: true
          title: Status
          value: '{{`{{ .Status }}`}}'
        - title: Summary
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.summary }}`}} {{`{{ end
}}`}}'
        - title: Description
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.description }}`}} {{`{{
end }}`}}'
        - title: Labels
          value: '{{`{{ range .Alerts }}`}} {{`{{ range .Labels.SortedPairs }}`}}{{`{{
printf "%s:
            %s\n" .Name .Value }}`}}{{`{{ end }}`}}{{`{{ end }}`}}'
        - title: Links
          value: '{{`{{ (index .Alerts 0).GeneratorURL }}`}}'
        title: '{{`{{ .CommonLabels.alertname }}`}}'
    route:
      groupBy:
      - '...'
      receiver: devnull
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[4-9]$"
          receiver: slack
      repeatInterval: 12h
  type: Internal
```

Example of sending alerts to Opsgenie

```
- name: opsgenie
      opsgenieConfigs:
        - apiKey:
            key: data
            name: opsgenie
          description: |
            {{ range .Alerts }}{{ .Annotations.summary }} {{ end }}
            {{ range .Alerts }}{{ .Annotations.description }} {{ end }}
          message: '{{ .CommonLabels.alertname }}'
          priority: P1
          responders:
            - id: team_id
              type: team
```

Example of sending alerts via email

Create a secret with the email account password. Specify the password encoded in Base64 format in the `password` field:

```
apiVersion: v1
kind: Secret
metadata:
  name: am-mail-server-pass
  namespace: d8-monitoring
data:
  password: BASE64_ENCODED_PASSWORD_HERE
```

Modify the values in the CustomAlertManager resource example according to the values relevant to your infrastructure and apply it:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: mail
spec:
  type: Internal
  internal:
    receivers:
      - name: devnull
      - name: mail
        emailConfigs:
          - to: oncall@example.com
            from: prom@example.com
            smarthost: mx.example.com:587
            authIdentity: prom@example.com
            authUsername: prom@example.com
            authPassword:
              key: password
              name: am-mail-server-pass
            # If you use a custom CA on the server, you can place the public part of
the CA in a ConfigMap in the d8-monitoring namespace
            # tlsConfig:
            #   insecureSkipVerify: true
            #   ca:
            #     configMap:
            #       key: ca.pem
            #       name: alertmanager-mail-server-ca
            sendResolved: true
            requireTLS: true
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: devnull
      repeatInterval: 24h
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[1-4]$"
          receiver: mail
```

Additional resources

- Module `prometheus` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Logging

Deckhouse Platform (DP) provides built-in tools for log collection, delivery, and short-term storage.

DP logging capabilities:

- Collect logs from cluster pods and nodes.
- Process logs, including metadata enrichment and message filtering.

- Deliver to various storage and analysis systems, including Loki, Elasticsearch, Splunk, and others.
- Short-term log storage in the cluster with search and visualization capabilities through Grafana.

The following sections describe how to configure log collection and delivery, as well as organize their short-term storage within the cluster.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Log collection and delivery

Deckhouse Platform (DP) provides log collection and delivery from cluster nodes and pods to internal or external storage systems.

DP allows you to:

- Collect logs from all or specific pods and namespaces.
- Filter logs by labels, message content, and other criteria.
- Send logs to multiple storage systems simultaneously (e.g., Loki and Elasticsearch).
- Enrich logs with Kubernetes metadata.
- Use log buffering to improve performance.

The general mechanism of log collection, delivery, and filtering is described in detail in the Architecture section.

DP administrators can configure log collection and delivery using three custom resources:

- ClusterLoggingConfig: Describes log sources at the cluster level, including collection, filtering, and parsing rules.
- PodLoggingConfig: Describes log sources within a specified namespace, including collection, filtering, and parsing rules.
- ClusterLogDestination: Defines log storage parameters.

Based on these resources, a *pipeline* is formed, which is used in DP to read logs and further work with them using the `log-shipper` module. A complete list of `log-shipper` module settings is available in the separate documentation section.

Configuring log collection and delivery

Below is a basic DP configuration option where logs from all cluster pods are sent to Elasticsearch-based storage.

To configure, follow these steps:

1. Enable the `log-shipper` module using the following command:

   ```
   d8 system module enable log-shipper
   ```

2. Create a ClusterLoggingConfig resource that defines log collection rules. This resource allows you to configure log collection from pods in a specific namespace and with specific labels, flexibly configure multi-line log parsing, and set other rules.

   In this example, it is specified that logs should be collected from all pods and sent to Elasticsearch:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLoggingConfig
   metadata:
     name: all-logs
   spec:
     type: KubernetesPods
     destinationRefs:
     - es-storage
   ```

3. Create a ClusterLogDestination resource, which describes the parameters for sending logs to

storage. This resource allows you to specify one or more storage systems and describe connection parameters, buffering, and additional labels that will be applied to logs before sending.

In this example, Elasticsearch is specified as the receiving storage:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Integration with external systems

You can configure DP to work with external log storage and analysis systems, such as Elasticsearch, Splunk, Logstash, and others, using the `type` parameter of the ClusterLogDestination resource.

Elasticsearch

To send logs to Elasticsearch, create a ClusterLogDestination resource following this example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Using index templates

To send messages to specific indices based on metadata using index templates, use the following configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s--%F"
```

In the example above, a separate index will be created in Elasticsearch for each Kubernetes namespace.

This feature is useful in combination with the `extraLabels` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s---%F"
  extraLabels:
    service: ""
```

- If the message has JSON format, the `service_name` field of this JSON document is moved to the metadata level.
- The new metadata field `service` is used in the index template.

Support for Elasticsearch < 6.X

To work with Elasticsearch versions prior to 6.0, enable support for `docType` indices using the ClusterLogDestination resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    docType: "myDocType" # Specify the value here. It should not start with '_'.
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Splunk

To configure sending events to Splunk, follow these steps:

1. Configure Splunk:
   - Define the endpoint. It should match your Splunk instance name with port `8088`, but without specifying the path, for example, `https://prd-p-xxxxxx.splunkcloud.com:8088`.
   - Create an access token. To do this, in Splunk, open the **Setting** -> **Data inputs** section, add a new **HTTP Event Collector** and copy the generated token.
   - Specify the Splunk index for storing logs, for example, `logs`.
2. Configure DP by adding a ClusterLogDestination resource to send logs to Splunk:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: splunk
spec:
  type: Splunk
  splunk:
    endpoint: https://prd-p-xxxxxx.splunkcloud.com:8088
    token: xxxx-xxxx-xxxx
    index: logs
    tls:
      verifyCertificate: false
      verifyHostname: false
```

> **i** `destination` does not support pod labels for indexing. To add the required labels, use the `extraLabels` option:
>
> ```
> extraLabels:
>  pod_label_app: ''
> ```

Logstash

To configure sending logs to Logstash, do the following:

1. Configure an incoming `tcp` stream with `json` codec on the Logstash side.

   Example Logstash configuration:

   ```
   input {
     tcp {
       port => 12345
       codec => json
     }
   }
   output {
     stdout { codec => json }
   }
   ```

2. Add a ClusterLogDestination resource:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLogDestination
   metadata:
     name: logstash
   spec:
     type: Logstash
     logstash:
       endpoint: logstash.default:12345
   ```

Graylog

To configure sending logs to Graylog, do the following:

1. Ensure that Graylog has an incoming stream configured to receive messages via TCP protocol on the specified port.
2. Create a ClusterLogDestination resource following the example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLogDestination
   metadata:
     name: test-socket2-dest
   spec:
     type: Socket
     socket:
       address: graylog.svc.cluster.local:9200
       mode: TCP
       encoding:
         codec: GELF
   ```

Message formats

You can choose the format of sent messages using the `.encoding.codec` parameter of the ClusterLogDestination resource:

- CEF

- GELF
- JSON
- Syslog
- Text

Below are configuration examples for some of them.

Syslog

Use the following configuration example to send messages via socket using TCP protocol in syslog format:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: rsyslog
spec:
  type: Socket
  socket:
    mode: TCP
    address: 192.168.0.1:3000
    encoding:
      codec: Syslog
  extraLabels:
    syslog.severity: "alert"
    # The request_id field must be present in the message.
    syslog.message_id: ""
```

CEF

DP can send logs in CEF format by using `codec: CEF`, with overriding `cef.name` and `cef.severity` based on values from the `message` field of the application log in JSON format.

In the example below, `app` and `log_level` are keys containing values for overriding:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: siem-kafka
spec:
  extraLabels:
    cef.name: ''
    cef.severity: ''
  type: Kafka
  kafka:
    bootstrapServers:
      - my-cluster-kafka-brokers.kafka:9092
    encoding:
      codec: CEF
    tls:
      verifyCertificate: false
      verifyHostname: true
    topic: logs
```

You can also set values manually:

```
extraLabels:
  cef.name: 'TestName'
  cef.severity: '1'
```

Log transformation

You can configure one or more types of transformations that will be applied to logs before sending to

storage.

Converting records to structured objects

The `ParseMessage` transformation allows you to convert a string in the `message` field to a structured JSON object based on one or more specified formats (String, Klog, SysLog, and others).

> ⚠ When using multiple `ParseMessage` transformations, string conversion (`sourceFormat: String`) must be performed last.

Example configuration for converting mixed format records:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
  - action: ParseMessage
    parseMessage:
      sourceFormat: JSON
  - action: ParseMessage
    parseMessage:
      sourceFormat: Klog
  - action: ParseMessage
    parseMessage:
      sourceFormat: String
      string:
        targetField: "text"
```

Example of the original log record:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
{"level" : { "severity": "info" },"msg" : "fetching.module.release"}
I0505 17:59:40.692994   28133 klog.go:70] hello from klog
```

Transformation result:

```
{... "message": {
  "text": "/docker-entrypoint.sh: Configuration complete; ready for start up"
  }
}
{... "message": {
  "level" : "{ "severity": "info" }",
  "msg" : "fetching.module.release"
  }
}
{... "message": {
  "file":"klog.go",
  "id":28133,
  "level":"info",
  "line":70,
  "message":"hello from klog",
  "timestamp":"2025-05-05T17:59:40.692994Z"
  }
}
```

Label replacement

The `ReplaceKeys` transformation allows you to recursively replace all matches of the `source` pattern

with the `target` value in the specified label keys.

> ⚠ Before applying the `ReplaceKeys` transformation to the `message` field or its nested fields, convert the log record to a structured object using the `ParseMessage` transformation.

Example configuration for replacing dots with underscores in labels:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: replace-dot
spec:
  ...
  transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
```

Example of the original log record:

```
{"msg" : "fetching.module.release"} # Pod label pod.app=test
```

Transformation result:

```
{... "message": {
  "msg" : "fetching.module.release"
  },
  "pod_labels": {
    "pod_app": "test"
  }
}
```

Label removal

The `DropLabels` transformation allows you to remove specified labels from a structured JSON message.

> ⚠ Before applying the `DropLabels` transformation to the `message` field or its nested fields, convert the log record to a structured object using the `ParseMessage` transformation.

Example configuration with label removal and preliminary `ParseMessage` transformation:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: DropLabels
      dropLabels:
        labels:
          - .message.example
```

Example of the original log record:

```
{"msg" : "fetching.module.release", "example": "test"}
```

Transformation result:

```
{... "message": {
  "msg" : "fetching.module.release"
  }
}
```

Log filtering

DP provides filters to exclude unnecessary messages to optimize the log collection process:

- `labelFilter` — applied to metadata, such as container name (`container`), namespace (`namespace`), or pod name (`pod_name`);
- `logFilter` — applied to message fields, if the message is in JSON format.

Collecting logs from a specific container

To configure filtering using `labelFilter`, create a ClusterLoggingConfig resource, using the configuration below as an example.

In this case, the filter selects logs from containers named `nginx`, and then sends them to internal Loki-based storage.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: nginx-logs
spec:
  type: KubernetesPods
  labelFilter:
  - field: container
    operator: In
    values: [nginx]
  destinationRefs:
  - loki-storage
```

Collecting logs without a specific string

Example configuration for collecting logs with filtering via `labelFilter`, where the `NotRegex` operator excludes strings matching the specified regular expression.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
  - loki-storage
  labelFilter:
  - field: message
    operator: NotRegex
    values:
    - .*GET /status" 200$
```

Kubelet audit events

Example configuration for collecting and filtering audit events related to kubelet operation, stored in the `/var/log/kube-audit/audit.log` file. Filtering is performed using `logFilter`, which searches for records in the `userAgent` field matching the regular expression `"kubelet.*"`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
  - field: userAgent
    operator: Regex
    values: ["kubelet.*"]
  destinationRefs:
  - loki-storage
```

DP system logs

Example configuration for collecting DP system logs located in the `/var/log/syslog` file. Message filtering using `labelFilter` allows you to select only those records that relate to the following components: `d8-kubelet-forker`, `containerd`, `bashible`, and `kernel`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: system-logs
spec:
  type: File
  file:
    include:
    - /var/log/syslog
  labelFilter:
  - field: message
    operator: Regex
    values:
    - .*d8-kubelet-forker.*
    - .*containerd.*
    - .*bashible.*
    - .*kernel.*
  destinationRefs:
  - loki-storage
```

> **i** If you need logs from only one pod or a small group of pods, use `kubernetesPods` to limit the collection scope. Filters should be used only for fine-tuning.

Log buffering

Using buffering improves the reliability and performance of the log collection system. Buffering can be useful in the following cases:

- **Temporary connection issues**. If there are temporary interruptions or connection instability with the log storage system (e.g., Elasticsearch), the buffer allows temporarily storing logs and sending them when the connection is restored.
- **Load spike smoothing**. During sudden log volume spikes, the buffer helps smooth the peak load on the storage system, preventing its overload and potential data loss.
- **Performance optimization**. Buffering helps optimize the performance of the log collection system by accumulating logs and sending them in batches, which reduces the number of network requests and improves overall throughput.

The `buffer` parameter of the ClusterLogDestination resource is responsible for configuring buffering.

Example of enabling in-memory buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    memory:
      maxEvents: 4096
    type: Memory
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of enabling disk buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of defining buffer overflow behavior

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
    whenFull: DropNewest
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Debugging and advanced features

Enabling debug logs for the log-shipper agent

To enable debug logs for the `log-shipper` agent on nodes with information about HTTP requests, connection reuse, tracing, and other data, enable the `debug` parameter in the `log-shipper` module configuration.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 1
  enabled: true
  settings:
    debug: true
```

Additional information about log transmission channels

Using Vector commands, you can get additional information about data transmission channels.

First, connect to one of the `log-shipper` pods:

```
d8 k -n d8-log-shipper get pods -o wide | grep $node
d8 k -n d8-log-shipper exec $pod -it -c vector -- bash
```

Execute subsequent commands from the pod’s command shell.

Topology overview

To get a diagram of your configuration topology:

1. Run the `vector graph` command. A diagram in DOT format will be generated.
2. Use WebGraphviz or a similar service to render the diagram based on the DOT file content.

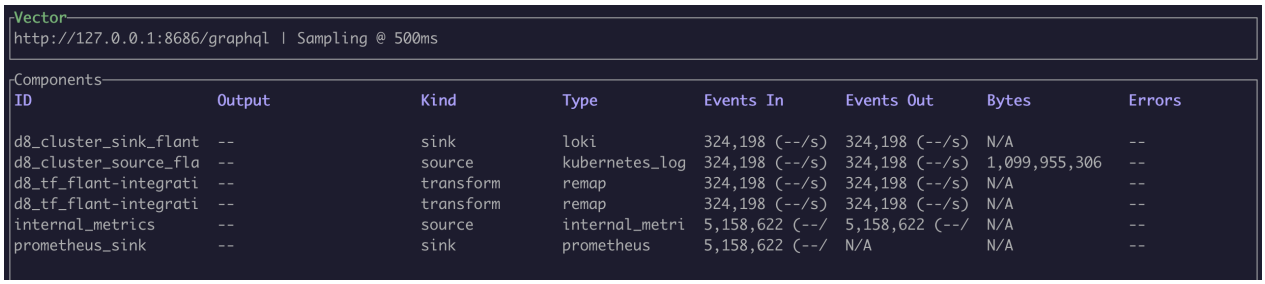
Example diagram for one log transmission channel in ASCII format:

```
+-----------------------------------------------+
|  d8_cluster_source_flant-integration-d8-logs  |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
|       d8_tf_flant-integration-d8-logs_0       |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
|       d8_tf_flant-integration-d8-logs_1       |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
| d8_cluster_sink_flant-integration-loki-storage |
+-----------------------------------------------+
```

Monitoring channel load

To view the traffic volume at each log processing stage, use the `vector top` command.

Example command output:



Getting raw and intermediate logs

To view input data at different log processing stages, use the `vector tap` command. By specifying the ID of a specific processing stage, you can see logs that arrive at that stage. Glob pattern selections are also supported, for example, `cluster_logging_config/*`.

Examples:

- Viewing logs before applying transformation rules (`cluster_logging_config/*` is the first processing stage according to the `vector graph` command output):

  ```
  vector tap 'cluster_logging_config/*'
  ```

- Modified logs entering the input of the next components in the channel chain:

  ```
  vector tap 'transform/*'
  ```

Debugging VRL rules

To debug rules in Vector Remap Language (VRL), use the `vector vrl` command.

Example VRL program:

```
. = {"test1": "lynx", "test2": "fox"}
del(.test2)
.
```

Adding support for new source or sink

The `log-shipper` module in DP is built based on Vector with a limited set of cargo features, to minimize the size of the executable file and speed up the build.

To view the complete list of supported features, run the `vector list` command.

If the required source or sink is missing, add the corresponding cargo feature to the Dockerfile.

Special cases

Collecting logs from production namespaces using the labelSelector option

If your cluster namespaces are labeled (e.g., `environment=production`), you can use the `labelSelector` option to collect logs from production namespaces.

Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: production-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchLabels:
          environment: production
  destinationRefs:
  - loki-storage
```

Label for excluding pods and namespaces

DP provides the `log-shipper.deckhouse.io/exclude=true` label for excluding specific pods and namespaces. It helps stop log collection from pods and namespaces without changing the global configuration.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    log-shipper.deckhouse.io/exclude: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
spec:
  ...
  template:
    metadata:
      labels:
        log-shipper.deckhouse.io/exclude: "true"
```

Additional resources

- Module `log-shipper` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Short-term log storage

Deckhouse Platform (DP) provides a built-in solution for short-term log storage based on the Grafana Loki project.

The storage is deployed in the cluster and integrated with the log collection system. After configuring ClusterLoggingConfig, PodLoggingConfig, and ClusterLogDestination resources, logs automatically flow from all system components. The configured storage is added to Grafana as a data source for visualization and analysis.

Log collection from user applications is configured separately.

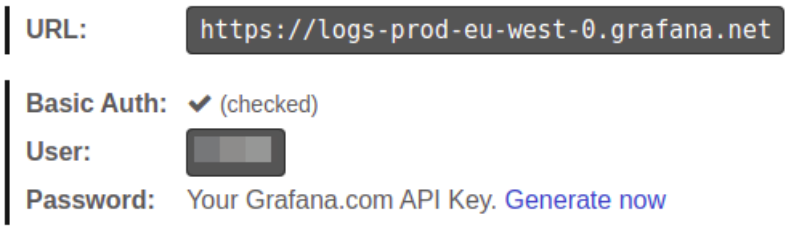
Short-term storage parameters are set in the `loki` module settings. You can configure disk size and retention period, specify the StorageClass to use, and set resource limits.

> ⚠ Short-term storage based on Grafana Loki does not support high availability mode. For long-term storage of important logs, use external storage.

Integration with Grafana Cloud

To configure DP to work with the Grafana Cloud platform, follow these steps:

1. Create a Grafana Cloud API access key.
2. Encode the Grafana Cloud access token in Base64 format:




```
echo -n "<YOUR-GRAFANACLOUD-TOKEN>" | base64 -w0
```

3. Create a ClusterLogDestination resource following the example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  loki:
    auth:
      password: PFlPVVItR1JBRkFOQUNMT1VELVRPS0VOPg==
      strategy: Basic
      user: "<YOUR-GRAFANACLOUD-USERNAME>"
    endpoint: <YOUR-GRAFANACLOUD-URL> # For example https://logs-prod-us-
central1.grafana.net or https://logs-prod-eu-west-0.grafana.net
  type: Loki
```

Migration from Grafana Promtail

To migrate from Promtail, edit the Loki URL by removing the `/loki/api/v1/push` path from it.

The Vector logging agent used in DP will automatically add this path when sending data to Loki.

Additional resources

- Module `log-shipper` custom resources
- Module `loki` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Multitenancy

Multitenancy is the ability to create isolated environments (projects) within a Kubernetes cluster. Projects are similar to namespaces but offer more capabilities. While namespaces are used for logical separation of resources in Kubernetes, they do not, for example, restrict network communication, pod resource consumption, or host directory mounts. These limitations make namespaces insufficient for modern development needs. By default, namespaces also do not include logging, auditing, and vulnerability scanning.

Using projects helps address these limitations and offers the following benefits:

- For platform administrators:
  - **Consistency**: Administrators can create projects using a shared template, which ensures consistency and simplifies management.
  - **Security**: Projects ensure isolation for resources and access policies between different tenants, supporting a secure multitenant environment.
  - **Resource consumption**: Administrators can easily set resource quotas and limits for each project to prevent resource overconsumption.
  - **Resource access management**: Administrators can control which cluster-wide resources (such as StorageClass, ClusterIssuer, ClusterRole, and LoadBalancerClass) each project may use, and set per-project default values. For details, refer to grants section in the usage guide.
- For platform users:
  - **Immediate start**: Developers can request projects created per templates from administrators to quickly start developing new applications.
  - **Isolation**: Each project provides an isolated environment, allowing developers to deploy and test their applications without affecting others.

> Secret copying across all namespaces is incompatible with projects in multitenancy mode.
>
> This mode creates isolated environments for users within their projects, while `secret-copier` automatically distributes secrets to all namespaces. If sensitive data is present in a user's private environment, it could lead to a data leak and a security model breach.

Additional namespaces

A project is not limited to a single namespace: if an application needs several namespaces (for example, a separate one for a cache or a queue), they can be added to the project as additional namespaces. The project's access grants and namespaced template policies (network isolation, log shipping) automatically apply to every namespace of the project, not just the main one.

Additional resources

- Module `secret-copier` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Project management

Deckhouse Platform includes a set of templates for creating projects. They are cumulative: each one includes the capabilities of the previous one and adds its own. Parameter values are set in the `.spec.parameters` field of a Project:

- `simple` is a minimal template that creates only the project namespace. Use it when you only need an isolated namespace managed as a project, while access and restrictions are configured through the standard project fields and project role bindings.

  Parameters:

  - `namespace.labels` and `namespace.annotations` — extra labels and annotations for the project namespace.
  - `requiredRequests` (default `false`) — rendered only if the parameter is declared in the template schema and is `true`. If `true`, the project pods must specify CPU and memory requests (an OperationPolicy in the `Deny` mode).

- `default` is a template for basic project use cases. On top of the namespace, it sets up network isolation, a pod security profile, extended monitoring, and log shipping.

  Parameters (in addition to the ones for `simple`):

  - `networkPolicy` — `Isolated` (default) allows incoming traffic only from the project namespaces, Prometheus, the ingress-nginx controller and the service-with-healthchecks agent, and outgoing traffic only to the project namespaces and to DNS in `kube-system`; `NotRestricted` allows all traffic.
  - `podSecurityProfile` — the Pod Security Standards profile for the project namespaces: `Baseline` (default) prevents known privilege escalations, `Restricted` applies the strictest hardening practices, `Privileged` restricts nothing. An empty string sets no profile: the project puts no `security.deckhouse.io/pod-policy` label on its namespaces, and the default policy of the `admission-policy-engine` module applies to them.
  - `extendedMonitoringEnabled` (default `true`) — alerts on controller outages and restarts, 5xx errors in ingress-nginx, and low free space on the project's persistent volumes.
  - `clusterLogDestinationName` — the name of the ClusterLogDestination to ship the project logs to. Left unset, the project logs are not shipped anywhere.
  - `requiredRequests` (default `true` in this template) — if `true`, the project pods must specify CPU and memory requests (an OperationPolicy in the `Deny` mode). When an existing namespace is adopted, the parameter is set to `false` so that the workloads already running are not blocked.

- `secure` includes all the capabilities of the `default` template, and additionally restricts the users and groups inside containers, audits their calls to the kernel, and scans images for vulnerabilities.

  Parameters (in addition to the ones for `default`):

  - `allowedUIDs` and `allowedGIDs` — the ranges (`min`, `max`) of IDs permitted for users and groups inside the project containers. See Security Context.
  - `runtimeAuditEnabled` (default `false`) — audit rules that catch calls to the kernel and find malicious activity. They work only if the UID/GID ranges are set.
  - `securityScanningEnabled` (default `true`) — periodic scanning of the launched images for known vulnerabilities (CVE) with Trivy, every 24 hours.

- `secure-with-dedicated-nodes` includes all the capabilities of the `secure` template, and additionally places the pods of the main namespace of the project on dedicated nodes.

  Parameters (in addition to the ones for `secure`). If `dedicatedNodes` is set, at least one of its two fields must be set. Without `dedicatedNodes`, the pods are not placed on dedicated nodes:

  - `dedicatedNodes.nodeSelector` — the node selector of the main namespace. It is merged into the node selector of every pod there, and a pod whose selector conflicts with it is rejected.
  - `dedicatedNodes.defaultTolerations` — tolerations in the format of the pod's `spec.tolerations`. They are **added** to the pods created in the main namespace.

The `default`, `secure`, and `secure-with-dedicated-nodes` templates are described in structured form (`deckhouse.io/v1alpha2`). The `simple` template is a minimal structured template that creates the namespace, sets its labels and annotations from the project parameters and, with `requiredRequests: true`, creates the OperationPolicy.

For the exact set of parameters, check the template installed in your cluster — it matches your platform version.

To view the parameter schema of a template, run the following command:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o
jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

To view the whole template, run the following command:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o yaml
```

Project creation

To create a project, follow these steps:

1. Create a custom resource Project with the project template name specified in the .spec.projectTemplateName field.
2. Set the standard fields — .spec.administrators and .spec.quota — which are managed directly by the Project resource regardless of the template.
3. In the .spec.parameters field, specify the values for the .spec.parametersSchema.openAPIV3Schema section of the custom resource ProjectTemplate.

   An example of creating a project using Project from the `default` ProjectTemplate is shown below:

   ```
   apiVersion: deckhouse.io/v1alpha3
   kind: Project
   metadata:
     name: my-project
   spec:
     description: This is an example from the Deckhouse documentation.
     projectTemplateName: default
     # Standard fields, managed by the Project resource itself, independently of
   the template.
     administrators:
       - kind: Group
         name: k8s-admins
     quota:
       requests.cpu: "5"
       requests.memory: 5Gi
       requests.storage: 1Gi
       limits.cpu: "5"
       limits.memory: 5Gi
     # Template-specific parameters.
     parameters:
       networkPolicy: Isolated
       podSecurityProfile: Restricted
       extendedMonitoringEnabled: true
   ```

> **i** The Project API is served as `deckhouse.io/v1alpha3`. `v1alpha2` manifests keep working: a conversion webhook automatically lifts `parameters.administrators` and `parameters.resourceQuota` into the `.spec.administrators` and `.spec.quota` standard fields. If one administrator does not fit `.spec.administrators` (for example, it has `accessLevel`), the whole list stays in `parameters`, and `.spec.administrators` is not filled. The same holds for a quota with anything other than quantities (for example, `scopes`), for a quota with the number `0` anywhere in it, and for an empty list or quota. The `deckhouse.io/v1alpha1` version is no longer served, and the API server refuses a manifest with `apiVersion: deckhouse.io/v1alpha1`. Before applying such a manifest, change `apiVersion` to `deckhouse.io/v1alpha3`, rename `spec.projectTypeName` to `spec.projectTemplateName` (a ProjectTemplate with that name must exist) and `spec.template` to `spec.parameters`, and move `administrators` and `resourceQuota` from the parameters to `.spec.administrators` and `.spec.quota`.

4. To check the project status, run the following command:

   ```
   d8 k get projects my-project
   ```

   A successfully created project should have the `Deployed` (synchronized) status. If the `Error` status is displayed, add the `-o yaml` flag to the command (for example, `d8 k get projects my-project -o yaml`) to get more details about the cause of the error.

Automatic project creation for a namespace

A namespace created directly (for example, `d8 k create ns test`) automatically becomes a project with the same name — no annotation is required:

- the template is picked from what the namespace already carries: `secure` if it has the `security-scanning.deckhouse.io/enabled` label, `default` if it has `security.deckhouse.io/pod-policy` or `extended-monitoring.deckhouse.io/enabled`, and `simple` otherwise;
- the project parameters are filled in from the current state of the namespace, so nothing inside it changes: the network policy stays unrestricted and the Pod Security Standard keeps the value the namespace already had;
- from then on the project is the source of truth and is edited like any other project. The namespace of the project cannot be deleted directly: the request is refused with the `Manual modification is forbidden` message. To remove the environment, delete the Project: `d8 k delete project <NAME>` deletes the namespace with it and returns once the namespace is gone. Only a user allowed to delete the namespace can delete its Project (see Additional project namespaces).

Users can create a namespace directly only while the `allowNamespacesWithoutProjects` parameter is `true` (the default). With `false`, only cluster administrators (`system:masters`, `system:sudouser`), Deckhouse, the module controller and system components create namespaces directly. Other users create a Project or a ProjectNamespace instead.

System namespaces (`d8-*`, `kube-*`, `upmeter-*`, `default`, and anything labeled `heritage: deckhouse` or `heritage: upmeter`) are never adopted this way: they belong to the virtual `deckhouse`/`default` projects (see Virtual projects). A namespace whose name is longer than 61 characters is not adopted either: that is the Project name limit.

A namespace that gets the `default` or `secure` template because of its monitoring or scanning label and has no `security.deckhouse.io/pod-policy` label gets `podSecurityProfile: ""`. The project puts no such label on it, and the default policy of the `admission-policy-engine` module keeps applying to it.

A namespace that already belongs to another Helm release (its `meta.helm.sh/release-name` annotation names a release other than the project's) is not taken over: the project gets the `HelmOwnership` condition with the name of that release and retries, and the namespace is left as it is. Remove the foreign release or its ownership annotations to let the project proceed.

Existing RoleBinding and AuthorizationRule objects inside the namespace keep working after adoption. The namespace Admin is **not** copied into `.spec.administrators` and does **not** become `d8:project:admin`. To make them a project administrator, add them to `.spec.administrators` or create a ProjectRoleBinding.

For example:

1. Create a new namespace:

   ```
   d8 k create ns test
   ```

2. Make sure the project has been created:

   ```
   d8 k get projects
   ```

   A new project matching the namespace will appear in the list of projects:

   ```
   NAME        STATE      PROJECT TEMPLATE   DESCRIPTION
   AGE
   deckhouse   Deployed   virtual            This is a virtual project
   181d
   default     Deployed   virtual            This is a virtual project
   181d
   test        Deployed   simple
   1m
   ```

To change the template of such a project, set another template in `.spec.projectTemplateName`.

⚠ If the namespace already has an object of the same kind and name as one that the new template creates, for example the `isolated` NetworkPolicy, the project switches to `Error`, and the `ResourcesUpgraded` condition names the object. Delete or rename the object first.

Creating a project without specifying a template

The `projectTemplateName` field is optional: when omitted, the `simple` template is used. Such a project consists only of the namespace and the standard fields (administrators, quota) — no template policies are created in it. This is convenient when no settings are needed or they are managed by other means:

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-plain-project
spec:
  administrators:
    - kind: Group
      name: k8s-admins
  quota:
    requests.cpu: "2"
```

A template can be assigned later by setting it in `.spec.projectTemplateName`.

Project naming rules

The project name is also the name of its main namespace, so the following rules are checked when a project is created:

- the name cannot start with `d8-` or `kube-` — these prefixes are reserved for system namespaces;
- the name cannot be `default` or `deckhouse` — these names belong to the virtual projects;
- the name cannot be longer than 61 characters;
- the name cannot be the name of an existing namespace, or of a namespace that a ProjectNamespace has claimed (the name is taken from the moment the ProjectNamespace is created, before the controller creates the namespace);
- the name cannot be the name of a namespace whose ServiceAccounts are still named among the subjects of a ClusterProjectRoleBinding, or of a ProjectRoleBinding of a project that the name extends with a dash and a suffix (`foo` for `foo-extra`). Such subjects are often left from a deleted additional namespace. Remove them from the binding first.

A project can be named after another project with a dash and a suffix (`foo-bar` beside `foo`). It is a separate project with a namespace of its own, not an additional namespace of `foo`, and the template, administrators, quota and bindings of `foo` do not apply to it. The webhook allows such a project with a warning, and the controller sends the `NamedLikeAdditionalNamespace` Warning event on the new project. The `MultitenancyManagerProjectNamedLikeAdditionalNamespace` alert lists every such pair of projects. To add a namespace to a project, create a ProjectNamespace.

Project status and diagnostics

The `.status.state` field of a project is either `Deployed` (all project resources are in sync, or the objects of `manifests` wait for their binding) or `Error`. The cause of an error is described in the conditions (`.status.conditions`):

```
d8 k get project my-project -o jsonpath='{range .status.conditions[*]}{.type}={.status}:
{.message}{"\n"}{end}'
```

Condition	False means
`ProjectTemplateFound`	The template referenced in `.spec.projectTemplateName` was not found.
`Validated`	The project parameters failed validation against the template schema (`parametersSchema`).
`ResourcesUpgraded`	The project resources could not be created or updated from the template (details in `message`).
`StandardFieldsApplied`	The standard fields (quota or administrators) could not be applied.
`TemplateRolesAllowed`	The template creates a binding to a role forbidden for granting in projects — the project switches to `Error`, the role is named in `message`.
`ManifestsApplied`	The objects of the template's `manifests` are not applied. The `message` says what they wait for, why none of them is applied, which existing objects they leave alone, or which objects the API server refused, with the code of each refusal (for example, `403 Forbidden`).
`HelmOwnership`	The namespace of the project belongs to another Helm release. The `message` names the release (see Automatic project creation for a namespace).
`NamespaceDeleted`	The project is being deleted, and its namespace is still terminating. The `message` says since when and what remains in it.

The `HandoverPending` condition is read the other way round. It is `True` while AuthorizationRule and ResourceQuota objects kept from a Helm text wait for the standard fields that replace them, and it is removed once they are deleted (see Switching a project from a Helm text).

Other useful status fields:

- `.status.namespaces` — all namespaces of the project with their kind (`Main`/`Additional`);
- `.status.usage` — the current quota usage (populated when `.spec.quota` is set);
- `.status.resources` — the state of the individual resources created from the template;
- `.status.keptObjects` — the objects of a Helm text that were left in place instead of being deleted;
- `.status.manifestObjects` — the objects applied from the template's `manifests`.

Service objects of a project

The controller creates service objects in the project namespaces. They are managed automatically — editing them manually is not possible (the attempt is rejected):

Object	Where	Comes from
`ResourceQuota/d8-project-quota`	The main namespace	The `.spec.quota` field of the project.
`ProjectRoleBinding/d8-administrators`	The main namespace	The `.spec.administrators` field of the project.

Object	Where	Comes from
`ProjectRoleBinding/d8-manifests`	The main namespace	The `manifests` field of the project template. It grants the `d8:project:admin` role to the user that applies these objects.
`RoleBinding/d8:prb:<name>`	Every namespace of the project	The fan-out of the ProjectRoleBinding named `<name>`.
`RoleBinding/d8:cprb:<name>`	Every namespace of every project	The fan-out of the ClusterProjectRoleBinding named `<name>`.

When the source object (a binding, the quota field, etc.) is removed, the corresponding service objects are removed automatically.

Virtual projects

Besides the user-created projects, the `d8 k get projects` list always contains two **virtual** projects (labeled `projects.deckhouse.io/virtual-project: "true"`):

- `deckhouse` — groups system namespaces (`d8-*`, `kube-*`, `upmeter-*`, `heritage: deckhouse` / `heritage: upmeter`);
- `default` — groups remaining namespaces that do not belong to any project (the `default` namespace itself).

Virtual-project status is rebuilt from the live namespace list: a deleted namespace disappears from that list. Virtual projects do not recreate namespaces.

Virtual projects exist for completeness: with them, every namespace of the cluster belongs to some project. They cannot be managed: they are not editable, ProjectNamespace and ProjectRoleBinding resources cannot be created in them, and ClusterProjectRoleBinding does not extend to them. Their project template, `virtual`, is reserved for them: a user project with `projectTemplateName: virtual` is refused by the webhook. No ClusterResourceGrantPolicy applies to their namespaces either, even one whose selectors match a virtual Project, because the cluster resource availability rules apply only in namespaces with the `projects.deckhouse.io/project` label, and the namespaces of virtual projects do not carry it.

No ProjectTemplate object named `virtual` exists, and the webhook refuses to create one. A template with this name that is already in the cluster is not used by any project and can be deleted.

Additional project namespaces

If an application needs several namespaces (for example, a separate one for a cache or a queue), add them to the project with the ProjectNamespace resource. It is created **in the main namespace of the project**; the resulting namespace is named `<project name>-<spec.name>`:

```
apiVersion: deckhouse.io/v1alpha3
kind: ProjectNamespace
metadata:
  name: cache
  namespace: my-project
spec:
  name: cache  # The my-project-cache namespace will be created.
```

You can check the project composition in its status:

```
d8 k get project my-project -o jsonpath='{.status.namespaces}'
```

The rules for working with ProjectNamespace:

- The `spec.name` field is immutable: to rename a namespace, delete the resource and create a new one.
- The resulting name `<project name>-<spec.name>` cannot be longer than 63 characters (the Kubernetes limit on namespace names).
- A ProjectNamespace can only be created in the main namespace of a project — it cannot be “nested” into an additional namespace or a foreign project.
- The resulting name must be free. The request is rejected when a namespace with that name

exists and is not a namespace of this project, when a project has that name, or when another ProjectNamespace already claims it (the ProjectNamespace `a-backend` of project `team` and the ProjectNamespace `backend` of project `team-a` both claim `team-a-backend`).
- The request is also rejected when the ServiceAccounts of a namespace with the resulting name are still named among the subjects of a ClusterProjectRoleBinding, or of a ProjectRoleBinding of another project that the resulting name extends with a dash and a suffix. Remove these subjects from the binding first.
- Deleting a ProjectNamespace resource deletes its namespace. Deleting the project deletes all of its namespaces.
- A project can be deleted only by a user allowed to delete each of its namespaces (the `delete` verb on `namespaces`). Otherwise the request is refused, and the refusal names the namespace the user may not delete.

What applies to the additional namespaces

The following automatically applies in **all** namespaces of the project (the main and the additional ones alike):

- **Access**: the ProjectRoleBinding and ClusterProjectRoleBinding bindings, including the automatic access of the project administrators. When a new namespace is added, all existing bindings fan out into it without any user action.
- **Namespaced template objects**: the network policy (`networkPolicy.mode: Isolated`) and the log collection setup (`logShipping`) are created in every namespace of the project. The network isolation allows traffic between the namespaces of one project. A Helm `resourcesTemplate` creates its objects in the main namespace unless an object names another namespace of the project. So does `manifests`, whose objects are not applied while one of them names a namespace outside the project.
- **Cluster-scoped template policies** (`OperationPolicy`, the `SecurityPolicy` from `allowedUIDs`/`allowedGIDs`): they select namespaces by the `projects.deckhouse.io/project` label, that is, they cover the whole project.
- **Inherited labels**: the pod security profile (`security.deckhouse.io/pod-policy`), extended monitoring (`extended-monitoring.deckhouse.io/enabled`), vulnerability scanning (`security-scanning.deckhouse.io/enabled`), the template label (`projects.deckhouse.io/project-template`) and the labels the template sets through `namespaceMetadata.labels` are synced from the main namespace to the additional ones. The sync is complete: if a feature is turned off or a label is removed in the template, the label is removed from the additional namespaces as well. The labels of the template are listed in the `projects.deckhouse.io/template-labels` annotation of each project namespace; they are changed through the Project or its ProjectTemplate, and a direct edit of one on an additional namespace is reverted. Thanks to the template label, the cluster resource availability rules also apply in all namespaces of the project.

The following stays in the **main** namespace only:

- the project quota (the `ResourceQuota` from `.spec.quota`);
- the annotations from the template's `namespaceMetadata`;
- the labels and annotations set on the main namespace directly or by a GitOps tool, including the ones the namespace had before it became a project, other than the pod security, monitoring and scanning labels above;
- the GitOps tracking labels (`app.kubernetes.io/instance`, `argocd.argoproj.io/*`, `kustomize.toolkit.fluxcd.io/*`, `helm.toolkit.fluxcd.io/*`, `kapp.k14s.io/*`), even when the template sets them;
- the node placement annotations (from the template's `nodeSelector` and `tolerations` fields).

When a namespace becomes a project automatically, its own labels and annotations are not copied into the Project, so the additional namespaces of the project do not inherit them. The pod security, monitoring and scanning labels are the exception: adoption turns them into parameters of the project. To put another label of such a namespace on the additional namespaces too, declare it in `spec.parameters.namespace.labels` of the Project, or set it on each additional namespace directly.

Labels of the project namespaces

Label	Main	Additional	Purpose
`projects.deckhouse.io/project: <project name>`	✓	✓	Project ownership — the common label of all namespaces of the project.

Label	Main	Additional	Purpose
`projects.deckhouse.io/project-namespace: <spec.name>`	—	✓	Marks an additional namespace (the name of the ProjectNamespace resource).
`projects.deckhouse.io/project-template: <template name>`	✓	✓	The project template; the cluster resource availability rules match by it.
`heritage: multitenancy-manager`	✓	✓	The namespace is managed by the project controller: its `spec`, finalizers and the labels listed in this table are changed through the Project; other labels and annotations may be changed directly.
`security.deckhouse.io/pod-policy`, `extended-monitoring.deckhouse.io/enabled`, `security-scanning.deckhouse.io/enabled`	✓	✓ (inherited)	Policies and features from the project template.
Labels from the template's `namespaceMetadata.labels`	✓	✓ (inherited)	Labels of the project template. They are listed in the `projects.deckhouse.io/template-labels` annotation of the namespace and changed through the Project or its ProjectTemplate.
Any other label set on a namespace directly or by a GitOps tool	—	—	Not set by the module. The project release neither reverts nor restores it, and the additional namespaces do not inherit it.

The common `projects.deckhouse.io/project` label makes it possible to select the project namespaces with a plain `get ns`.

To get all namespaces of the project (main + additional):

```
d8 k get ns -l projects.deckhouse.io/project=my-project
```

To get additional namespaces only:

```
d8 k get ns -l 'projects.deckhouse.io/project=my-
project,projects.deckhouse.io/project-namespace'
```

To get the main namespace only:

```
d8 k get ns -l 'projects.deckhouse.io/project=my-
project,!projects.deckhouse.io/project-namespace'
```

Standard project fields

Project administrators and resource quotas are not project template parameters — they are top-level fields of the Project resource and work with any template (including `simple`):

- `.spec.administrators` — a list of subjects (`kind: User` or `kind: Group` and `name`) that receive administrative access to the project. The controller implements this access through an auto-generated ProjectRoleBinding named `d8-administrators` in the project's main namespace. A name cannot start with `system:multitenancy-manager:`, the prefix of the users that the controller acts as.
- `.spec.quota` — a set of hard ResourceQuota limits (for example, `requests.cpu`, `limits.memory`). The controller maintains a `ResourceQuota` in the project's main namespace and reports current usage in `.status.usage`. For `memory` and `storage`, a unit is required (for example, `2Gi`). Numbers without a unit mean bytes, and all of them except 0 are rejected. An update that leaves such a value as it was is accepted with a warning, and so is a create through `deckhouse.io/v1alpha2` with such numbers in `parameters.resourceQuota`.

⚠ A Helm `resourcesTemplate` that renders a ResourceQuota or an AuthorizationRule keeps rendering it, next to the ResourceQuota and the ProjectRoleBinding that the standard fields create. A pod in such a project has to fit every ResourceQuota of its namespace, and the access such AuthorizationRules grant adds to the access of the standard fields.

Granting access within a project

To grant access to project namespaces for users beyond the project administrators, use role bindings that reference cluster-wide roles and fan out into the appropriate project namespaces automatically:

- ProjectRoleBinding (namespaced, short name `prb`) — grants a role within a **single** project. It must be created in the project's main namespace. The controller creates a RoleBinding in every namespace of that project.
- ClusterProjectRoleBinding (cluster-scoped, short name `cprb`) — grants a role across **all** non-virtual projects.

`roleRef` must reference a `ClusterRole` whose name starts with one of the allowed prefixes (`d8:project:`, `d8:namespace:`, `d8:project-capability:`, `d8:namespace-capability:`, `d8:custom:`). See the user-authz module documentation for the description of the roles.

The following checks apply when a binding is created:

- **Privilege escalation protection**: a binding can only be created by a user who has the right to bind (`bind`) the referenced role; the right is checked by the role's name. A project administrator (`d8:project:admin`) has `bind` on exactly eight roles: `d8:project:viewer`, `d8:project:user`, `d8:project:manager`, `d8:project:admin`, `d8:namespace:viewer`, `d8:namespace:user`, `d8:namespace:manager`, and `d8:namespace:admin`. A binding to any other role — a custom `d8:custom:*` role, a capability role, or `d8:project:superadmin` — is rejected for a project administrator, even if it is within their own permissions, because nobody granted them `bind` on that name. Such bindings are created by a cluster administrator, or the cluster administrator grants `bind` on that role to project administrators separately via a dedicated ClusterRole.
- The role must exist: a binding to a non-existent role is rejected.
- A ServiceAccount used as a ProjectRoleBinding subject must belong to a namespace of that same project: its main namespace, its additional namespaces or a namespace a ProjectNamespace of the project has claimed. A namespace that is only named like an additional namespace (`<project>-*`) but belongs to a separate project does not qualify.
- System and subsystem roles (`d8:system:*`, `d8:subsystem:*`), as well as arbitrary roles outside the listed prefixes, cannot be granted through project bindings.
- Roles with the `rbac.deckhouse.io/disabled-for-direct-use-in-projects: "true"` annotation are forbidden for granting in projects. A cluster administrator can put this annotation on a role to stop new grants of it: existing bindings keep working, but no new ones are created. If a project template uses such a role, the project switches to `Error` with the reason in the `TemplateRolesAllowed` condition.

The controller checks the ServiceAccounts by the same rule every time it updates the RoleBindings of a ProjectRoleBinding, and also when a namespace appears, is deleted or moves to another project. The subjects that do not pass stay in `.spec.subjects` but get no rights: the `SubjectsBound` condition of the binding is `False` and lists them. Once the namespace is a namespace of the project again, the subjects get the rights back without a change to the binding.

The `d8-administrators` binding, created by the controller from the `.spec.administrators` field, is managed by the controller only — it cannot be edited manually. To change the set of administrators, change the project's `.spec.administrators` field instead. The ServiceAccount rule above does not apply to this binding.

The following bindings give the `viewer@example.com` user the `d8:project:viewer` role in every

namespace of `my-project`, and the `platform` group the same role in every project except the virtual ones:

```
---
apiVersion: deckhouse.io/v1alpha3
kind: ProjectRoleBinding
metadata:
  name: viewers
  namespace: my-project
spec:
  subjects:
    - kind: User
      name: viewer@example.com
  roleRef:
    kind: ClusterRole
    name: d8:project:viewer
---
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: platform-viewers
spec:
  subjects:
    - kind: Group
      name: platform
  roleRef:
    kind: ClusterRole
    name: d8:project:viewer
```

Roles available in a RoleBinding inside a project

Besides the project bindings, a plain RoleBinding can also be used inside a project namespace — the role then applies in that single namespace only. However, in projects the set of roles available to a plain RoleBinding is restricted: only cluster roles carrying the `rbac.deckhouse.io/delegatable: "true"` label are allowed. Among the built-in ones these are the `d8:namespace:*` and `d8:project:*` roles, as well as the access-level roles of the basic role model (`user-authz:user`, `user-authz:privileged-user`, `user-authz:editor`, `user-authz:admin`).

A RoleBinding to any other cluster role (for example, `cluster-admin`, system roles, or capabilities) is rejected in a project with the message `references "<role>" which is not available to project`. This protects the project isolation from being bypassed by binding to an overly broad role.

To use a custom role in projects, add the `rbac.deckhouse.io/delegatable: "true"` label to it:

```
d8 k label clusterrole d8:custom:namespace:developer rbac.deckhouse.io/delegatable=true
```

The restriction applies in the namespaces of every project, including the ones adopted from a standalone namespace.

Structured templates

Starting with the `deckhouse.io/v1alpha2` API version, a project template is described by **structured fields** — instead of a text Helm template, you declaratively specify which settings the project namespaces get. The controller itself creates the corresponding objects (network policies, security policies, log collection settings, etc.) from these fields in every namespace of the project and keeps them up to date. Other objects of the project namespaces are described in `manifests`.

Available fields (all optional; the complete reference is in the ProjectTemplate resource description):

Field	What it configures
`podSecurityStandard`	Pod security profile: `Privileged`, `Baseline`, or `Restricted`.

Field	What it configures
`networkPolicy.mode`	Network isolation: `Isolated` (traffic is only allowed within the project and from the platform system components) or `NotRestricted`.
`features.monitoring`	Extended monitoring of the project namespaces.
`features.vulnerabilityScanning`	Scanning of container images for vulnerabilities.
`logShipping.clusterDestinationRef`	Collecting the logs of the project pods into the given destination (`ClusterLogDestination`).
`nodeSelector`, `tolerations`	Placing the pods of the main namespace on dedicated nodes.
`allowedUIDs`, `allowedGIDs`	The allowed UID/GID ranges of the project containers.
`runtimeAudit.enabled`	Auditing the project processes' access to the Linux kernel.
`namespaceMetadata.labels`, `namespaceMetadata.annotations`	Extra labels and annotations of the project namespaces. The labels the module manages cannot be set here: a template or a Project that sets one is refused.
`resources`, `grantPolicies`	Granting cluster-scoped resources through a project template — see Managing access to cluster-wide resources.
`manifests`	Other objects in the project namespaces, applied with the rights of a project administrator. See Describing other objects in manifests.
`parametersSchema.openAPIV3Schema`	The schema of parameters set when creating a project.

An example of a structured template:

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: my-template
spec:
  title: "Team template"
  description: "Isolated project with monitoring"
  podSecurityStandard: Baseline
  networkPolicy:
    mode: Isolated
  features:
    monitoring: true
    vulnerabilityScanning: true
```

Template parametrization

Twelve fields of the template can be turned into a parameter: `podSecurityStandard`, `networkPolicy.mode`, `features.monitoring`, `features.vulnerabilityScanning`, `logShipping.clusterDestinationRef`, `nodeSelector`, `tolerations`, `allowedUIDs`, `allowedGIDs`, `runtimeAudit.enabled`, `namespaceMetadata.labels` and `namespaceMetadata.annotations`. Instead of a concrete value, specify `{fromParam: <parameter name>}` and declare the parameter in `parametersSchema`. The value does not have to be a scalar: a map (`nodeSelector`, `namespaceMetadata.labels`), a list (`tolerations`) or an object (`allowedUIDs`) works just as well. The other fields — `title`, `description`, `resources`, `grantPolicies` and `parametersSchema` itself — take literal values only. Each project then sets its own value in `.spec.parameters`; if the value is not set, the `default` from the schema is used.

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: my-parametrized-template
spec:
  podSecurityStandard:
    fromParam: securityProfile
  networkPolicy:
    mode:
      fromParam: networkMode
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        securityProfile:
          type: string
          enum: [Baseline, Restricted]
          default: Baseline
        networkMode:
          type: string
          enum: [Isolated, NotRestricted]
          default: Isolated
```

A project using such a template:

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-project
spec:
  projectTemplateName: my-parametrized-template
  parameters:
    securityProfile: Restricted
```

The `fromParam` references are checked when a template is created or changed: a reference to an undeclared parameter or to a parameter of an incompatible type (for example, a string parameter for a boolean field) is rejected.

The `requiredRequests` parameter is read by name, without `fromParam`. The controller renders an OperationPolicy in the `Deny` mode that requires CPU and memory requests from the pods in every namespace of the project only when the template declares the parameter in `parametersSchema` with `type: boolean` and its value for the project is `true`. A template that does not declare the parameter renders no such policy, even if the project sets `requiredRequests` as an undeclared key.

Template checks

The following rules apply to template operations:

- A template cannot be deleted while at least one project uses it or a project namespace is still rendered from it. After a project switches to another template, the old template stays until the new one has rendered.
- A change to a template is automatically applied to all projects created from it.
- An existing template can describe its objects with a Helm text in `resourcesTemplate` instead of structured fields. This form is deprecated, and the `resourcesTemplate` field will be removed together with `deckhouse.io/v1alpha1` in one of the upcoming releases. The `MultitenancyManagerHelmProjectTemplate` alert names every template that still renders from a Helm text, except the built-in ones. Describe new templates with structured fields and `manifests`, and move the existing ones as Moving a template from a Helm text describes.
- A template cannot get a new `resourcesTemplate` through either version. A create with a non-empty `resourcesTemplate` and an update that sets it on a template whose `resourcesTemplate` is empty are refused. A template that has a Helm text keeps it, and the text can be changed through either version. So a template with a Helm text cannot be created again, for example from a manifest exported from another cluster, from a backup, or from Git after the template was deleted. Move such a template to structured fields and `manifests` first.
- The `deckhouse.io/v1alpha1` version of ProjectTemplate is served and deprecated, and the API server returns a warning on every request through it. A template is created through

`deckhouse.io/v1alpha2` only, and a create through `deckhouse.io/v1alpha1` is refused. The templates created through `deckhouse.io/v1alpha1` earlier are read, changed and deleted through it as before. A change through `v1alpha1` to a template that has fields `v1alpha1` cannot describe, such as `title`, `grantPolicies`, `podSecurityStandard` or `manifests`, is refused, because it would erase them. Change such a template through `deckhouse.io/v1alpha2`.

Creating a custom project template

The default project templates cover common baseline scenarios and also serve as examples of what templates can do.

To create your own template:

1. Use one of the default templates as a starting point, for example, `default`.
2. Export it to a separate file, for example, `my-project-template.yaml`, using the following command:

   ```
   d8 k get projecttemplates default -o yaml > my-project-template.yaml
   ```

3. Edit the `my-project-template.yaml` file: adjust the structured fields and the input parameters schema to your needs.
4. Change the template name in the `.metadata.name` field.
5. Remove the `heritage` label, the `metadata.resourceVersion`, `metadata.uid`, `metadata.creationTimestamp`, `metadata.generation` and `metadata.managedFields` fields and the `status` section from the file.
6. Apply the resulting template with the following command:

   ```
   d8 k apply -f my-project-template.yaml
   ```

7. Check that the new template is available by running:

   ```
   d8 k get projecttemplates <NEW_TEMPLATE_NAME>
   ```

Labels for the objects of a Helm text

The following labels change how the module handles the objects of a Helm text in `resourcesTemplate`. Only existing templates have such a text, because no template can get a new `resourcesTemplate`. The objects of `manifests` never get the `heritage: multitenancy-manager` label, and the module does not read these labels on them.

Skipping the heritage label

By default, every object of the Helm text gets the `heritage: multitenancy-manager` label. This label prohibits changes to the object by users or any other controller except `multitenancy-manager`. To allow changes to the object (for example, for compatibility with other systems, or to control the created objects yourself), add the `projects.deckhouse.io/skip-heritage-label` label to it.

Example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/skip-heritage-label: "true"
    app: my-app
data:
  key: value
```

In this case, the object gets the `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` labels, but does not get the `heritage: multitenancy-manager` label.

Excluding objects from management

To exclude an object from management by the module (for example, when it is managed manually or by another controller), add the `projects.deckhouse.io/unmanaged` label to it.

Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Objects with the `projects.deckhouse.io/unmanaged` label:

- are created **only once**, when the project release is first installed, so an object added to the template later is not created in existing projects;
- **are not updated** when the template changes;
- are not tracked in the project status;
- get the `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` labels, but **not** the `heritage: multitenancy-manager` label;
- get the `helm.sh/resource-policy: keep` annotation, so the project release does not delete them.

⚠ After it is created, an `unmanaged` object does not depend on the template and must be managed manually.

Implementing validation of object changes with a custom label

The `multitenancy-manager` module uses ValidatingAdmissionPolicy to protect resources labeled `heritage: multitenancy-manager` from manual changes.
You can implement similar validation for resources with any label.

How validation works in multitenancy-manager

`multitenancy-manager` validates objects labeled `heritage: multitenancy-manager`.
The following resources are used for this:

1. ValidatingAdmissionPolicy: Defines validation rules:
   - Operations: `CREATE`, `UPDATE` and `DELETE`.
   - Exclusions: requests from cluster components are not checked, that is, from system groups (`system:nodes`, the `kube-system` and `d8-system` service accounts) and system users (`system:apiserver`, `system:kube-controller-manager`, `system:kube-scheduler` and others).
   - Check: only operations on behalf of the controller's service account are allowed.
   - Applies to all resources and API groups.
2. ValidatingAdmissionPolicyBinding: Defines which objects the validation applies to:
   - Uses `namespaceSelector` and `objectSelector` to select resources by the label `heritage: multitenancy-manager`.

Creating your own validation

To implement validation for resources with a different label (for example, `heritage: my-custom-label`):

1. Create a file with the ValidatingAdmissionPolicy and ValidatingAdmissionPolicyBinding resource manifests:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:   ["*"]
        apiVersions: ["*"]
        operations:  ["CREATE", "UPDATE", "DELETE"]
        resources:   ["*"]
        scope: "*"
  matchConditions:
    # Let the cluster components through, as the policy of the module does.
    - name: exclude-system-groups
      expression: '!(["system:nodes", "system:serviceaccounts:kube-system",
"system:serviceaccounts:d8-system"].exists(e, (e in request.userInfo.groups)))'
    - name: exclude-system-users
      expression: '!(["system:apiserver", "system:kube-controller-manager",
"system:kube-scheduler"].exists(e, (e == request.userInfo.username)))'
  variables:
    # A DELETE request carries no object, only oldObject.
    - name: subject
      expression: 'object == null ? oldObject : object'
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-
namespace:my-service-account"' # Replace with your ServiceAccount.
      reason: Forbidden
      messageExpression: 'variables.subject.kind == ''Namespace'' ? ''This
resource is managed by '' + variables.subject.metadata.name + '' system. Manual
modification is forbidden.''
       : ''This resource is managed by '' + variables.subject.metadata.namespace
+ '' system. Manual modification is forbidden.'''
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    objectSelector:
      matchLabels:
        heritage: my-custom-label
```

2. Configure the validation parameters:
   - `policyName`: Unique policy name (must match in Policy and Binding).
   - `request.userInfo.username`: The name of the ServiceAccount allowed to change objects (replace with your ServiceAccount).
   - `heritage: my-custom-label`: The value of the `heritage` label for your resources (replace with your value). The use of the values `multitenancy-manager`, `deckhouse` is prohibited.
   - `failurePolicy`: What happens when the policy cannot be evaluated, for example when an expression fails: `Fail` rejects the request, `Ignore` admits it. A validation that returns `false` rejects the request with the `Deny` action in any case.
   - `matchConditions`: Requests that the policy does not check. Keep the cluster components in this list, otherwise the policy blocks the deletion of namespaces and the garbage collection of the labeled objects.
   - `validationActions`: Validation actions:
     - `Deny`: Deny unauthorized operations.
     - `Audit`: Record operations in the audit log.

3. Apply the policy:

3. Apply the policy:

```
d8 k apply -f my-validation-policy.yaml
```

4. Ensure your resources have the corresponding `heritage` label:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label
```

Managing access to cluster-wide resources

The `multitenancy-manager` module allows cluster administrators to define, for each project, which cluster-wide resources (such as StorageClass, ClusterIssuer, ClusterRole, and LoadBalancerClass) can be used from project namespaces and which values are used by default.

This mechanism works independently of RBAC. RBAC determines *who can create and modify* objects, while the cluster-wide resource access mechanism determines *which resources* those objects can use.

The mechanism uses four custom resources:

- GrantableClusterResourceDefinition registers a type of cluster-wide resource whose access can be managed. These resources are provided by DP or module developers.
- GrantableClusterResourceReference defines where a registered cluster-wide resource is used, for example, which resource field contains a reference to it. These resources are provided by modules.
- ClusterResourceGrantPolicy defines access rules: the allowed and denied resources, as well as the resource used by default. A cluster administrator selects the namespaces to which the policy applies with two selectors: `projectSelector` matches the labels of the Project, and `namespaceSelector` matches the labels of the Namespace. A policy with neither selector is a library policy: it applies to a project only through `grantPolicies` of its ProjectTemplate.
- AvailableClusterResource is a read-only list of cluster-wide resources available to the project under the policies that apply. The controller keeps one in every namespace of every project for each GrantableClusterResourceDefinition.

Until the administrator creates a ClusterResourceGrantPolicy, resource availability is determined by their registration: resources are available to all projects if `defaultAvailability: All` (the default value) is set in GrantableClusterResourceDefinition and the resource does not match the `excluded` or `reserved` filters. Access checks are performed only for objects in project namespaces. If an access policy changes, cluster-wide resources already used by existing objects remain available to those objects.

The webhooks reject requests while the module controller is unavailable. Until it is back, objects of the registered kinds (for example, PersistentVolumeClaim, Service or RoleBinding) cannot be created or changed in project namespaces, except by the platform components and members of the `system:masters` group. To keep the controller available, enable `highAvailability`.

For a detailed description of the access management mechanism, refer to the `multitenancy-manager` module documentation.

Additional resources

- Module `admission-policy-engine` parameters
- Module `multitenancy-manager` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Overview

Managed services in Deckhouse Platform let you run ready-to-use application services in a cluster, such as databases, message queues, and big data services, without having to manage their lifecycle manually. DP handles the technical tasks of administration, scaling, backup, and updates, while the administrator manages managed services at the level of classes, limits, and architectural decisions.

How it works

Each managed service adds its own Kubernetes resources and controllers to the cluster. A user describes the desired state of the managed service in a namespaced resource. The controller creates and updates the related objects: workload instances, Services, secrets, snapshots, and other resources supported by the specific managed service. The administrator manages cluster-wide resources that define allowed configurations and default values for user resources.

Use this section to:

- prepare the cluster to work with managed services;
- configure shared settings and limits;
- define available managed service classes and configurations;
- verify that the infrastructure is ready to run managed services.

For PostgreSQL administration, see Managed PostgreSQL.

Instructions for creating managed services, configuring instances, and connecting to them are provided in the Usage section.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed PostgreSQL

The Managed PostgreSQL service helps you deploy and maintain PostgreSQL instances in DP. Supported PostgreSQL version — 17.6.

This page describes the settings a cluster administrator can configure: managing resource limits and topology, configuring parameter validation, setting default values, and binding to cluster nodes.

Enabling the service

To enable the Managed PostgreSQL service in DP, complete the following steps:

1. Make sure the requirements for the `managed-postgres` module are met.
2. Enable the `managed-postgres` module in the DP web interface or another way.
3. Create a PostgresClass with the settings that users will need when creating Postgres objects. You can also use the `default` PostgresClass that the module creates when enabled. The `default` PostgresClass contains baseline settings so users can start creating databases right after Managed PostgreSQL is enabled in DP. For production environments, it’s recommended to create separate PostgresClass resources with explicit settings and limits.

Once the module is enabled, users can create PostgreSQL databases on their own. User operations with the service are described in Usage → Managed services → Managed PostgreSQL.

Dependencies for specific features

Some module features require additional configuration of Deckhouse Platform components or cluster infrastructure:

Feature	Requirement
Backup (managed by PostgresSnapshot)	The snapshot-controller module enabled and a StorageClass with snapshot support
TLS via `cert-manager`	The cert-manager module enabled and a configured ClusterIssuer or Issuer

Feature	Requirement
Placement on dedicated nodes	Node labels (for example, `node.deckhouse.io/group=pg`) and taints if needed. For more information, see the documentation

Example of creating a PostgresClass

Before applying the example, make sure the cluster has enough resources. To estimate the available resources, run:

```
d8 k describe node worker-0 | grep -A 5 "Allocated resources"
```

Example output:

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                3176m (80%)       500m (12%)
  memory             8342837084 (71%)  6400Mi (57%)
```

Below is an example of a `production-v1` PostgresClass that you can use instead of the `default` PostgresClass and adapt to your requirements for resources, topology, PostgreSQL parameters, and node placement. The following sections describe these parameters in more detail.

Create a `postgresclass-production.yaml` file with the following content:

```
apiVersion: managed-services.deckhouse.io/v1alpha1
kind: PostgresClass
metadata:
  name: production-v1
spec:
  # Limit CPU and memory resources.
  sizingPolicies:
    - cores:
        min: 1
        max: 2
      memory:
        min: 512Mi
        max: 2Gi
        step: 512Mi
      coreFractions:
        - 50
        - 100
    - cores:
        min: 3
        max: 4
      memory:
        min: 2Gi
        max: 4Gi
        step: 1Gi
      coreFractions:
        - 50
        - 100

  # Manage fault tolerance.
  topology:
    allowedTopologies:
      - Ignored
      - Zonal
      - TransZonal
```

```
    defaultTopology: TransZonal
    allowedZones:
      - zone-a
      - zone-b
      - zone-c

  # Validate PostgreSQL settings.
  validations:
    - message: "Max connections should not be more than 300"
      rule: "configuration.maxConnections <= 300"
    - message: "Shared buffers should not be more than 25% of RAM"
      rule: "configuration.sharedBuffers < instance.memory.size / 4"
    - message: "walKeepSize can not be more than 1Gi"
      rule: "configuration.walKeepSize <= 1073741824"

  # Default values and override permissions.
  configuration:
    maxConnections: 200
    sharedBuffers: 1Gi
  overridableConfiguration:
    - maxConnections
    - sharedBuffers
    - walKeepSize

  # Bind to dedicated nodes.
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
          - key: "node.deckhouse.io/group"
            operator: "In"
            values:
              - "pg"
              - "postgres"
  nodeSelector:
    "node.deckhouse.io/group": "pg"
  tolerations:
    - key: primary-role
      operator: Equal
      value: pg
      effect: NoSchedule
```

Apply the manifest:

```
d8 k apply -f postgresclass-production.yaml
```

Check the created PostgresClass:

```
d8 k describe postgresclass production-v1
```

Example output...

```
Name:         production-v1
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  managed-services.deckhouse.io/v1alpha1
Kind:         PostgresClass
Metadata:
  Creation Timestamp:  2026-08-06T13:08:24Z
  Generation:          1
  Resource Version:    21610964
```

```
  UID:                f2548e11-a23c-4bd1-b786-776bf1677a1d
Spec:
  Configuration:
    Max Connections:  200
    Shared Buffers:   1Gi
  Node Affinity:
    Required During Scheduling Ignored During Execution:
      Node Selector Terms:
        Match Expressions:
          Key:       node.deckhouse.io/group
          Operator:  In
          Values:
            pg
            postgres
  Node Selector:
    node.deckhouse.io/group:  pg
  Overridable Configuration:
    maxConnections
    sharedBuffers
    walKeepSize
  Sizing Policies:
    Core Fractions:
      50
      100
    Cores:
      Max:  2
      Min:  1
    Memory:
      Max:   2Gi
      Min:   512Mi
      Step:  512Mi
    Core Fractions:
      50
      100
    Cores:
      Max:  4
      Min:  3
    Memory:
      Max:   4Gi
      Min:   2Gi
      Step:  1Gi
  Tolerations:
    Effect:    NoSchedule
    Key:       primary-role
    Operator:  Equal
    Value:     pg
  Topology:
    Allowed Topologies:
      Ignored
      Zonal
      TransZonal
    Allowed Zones:
      zone-a
      zone-b
      zone-c
    Default Topology:  TransZonal
  Validations:
    Message:  Max connections should not be more than 300
    Rule:     configuration.maxConnections <= 300
    Message:  Shared buffers should not be more than 25% of RAM
    Rule:     configuration.sharedBuffers < instance.memory.size / 4
    Message:  walKeepSize can not be more than 1Gi
```

```
    Rule:     configuration.walKeepSize <= 1073741824
Events:       <none>
```

Users can use the created PostgresClass by specifying it in the postgresClassName parameter of Postgres.

⚠ Once a PostgresClass is created, its parameters can't be changed. Create a new PostgresClass with a different name to use different settings.

Change and delete a PostgresClass

Before deleting an old class, check whether it's used by any existing Postgres objects. Run:

```
d8 k get postgres --all-namespaces -o custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,CLASS:.spec.postgresClassName | grep production-v1
```

If Postgres objects using this PostgresClass are found, keep the following in mind:

- Databases created based on the deleted PostgresClass keep running. Their settings stay the same, since they were fixed at creation time.
- Users won't be able to create a new Postgres object that references the deleted class.

If the PostgresClass is no longer used, you can delete it.

Example of deleting the `production-v1` PostgresClass via the CLI:

```
d8 k delete postgresclass production-v1
```

Example output:

```
postgresclass.managed-services.deckhouse.io "production-v1" deleted
```

i If you need to change PostgresClass settings, it's recommended to create a new one with a different name rather than recreating a PostgresClass with the same name.

Limit CPU and memory resources

Policies, in the `spec.sizingPolicies` parameter section, are used to define the allowed CPU and memory combinations for PostgreSQL instances. The administrator can set several core ranges, each with a minimum and maximum amount of memory and a step. These policies are useful when several independent teams work in the same cluster and create Postgres objects without coordinating with the administrator. Policies prevent Postgres objects with unrealistic resource requests and ensure predictable node utilization.

The policy is selected by the number of CPU cores. The `coreFractions` parameter defines what percentage of the CPU limits (`limits`) becomes the guaranteed request (`requests`). For example, if the administrator specifies `coreFractions: [50, 100]`, a user creating a Postgres object can choose `coreFraction: 50` or `coreFraction: 100`.

With `coreFraction: 50`:

- The CPU `limits` equal the number of requested cores (for example, 4 cores).
- The CPU `requests` are 50% of `limits` (2 cores).

The scheduler guarantees the pod 2 cores but allows it to use up to 4 if they're available. This increases pod placement density on nodes.

With `coreFraction: 100`:

- `limits` and `requests` are equal (4 cores).
- The pod gets a guaranteed resource allocation, but loses the ability to reuse unused cores from other pods.

Below is a fragment of a PostgresClass manifest showing the allowed CPU and memory combinations. The complete example is provided in Example of creating a PostgresClass.

```
spec:
  sizingPolicies:
    - cores:
        min: 1
        max: 2
      memory:
        min: 512Mi
        max: 2Gi
        step: 512Mi
      coreFractions:
        - 50
        - 100
```

In this fragment, the available memory values for 1–2 cores are 512Mi, 1Gi, 1.5Gi, and 2Gi.

The `step` parameter defines the increment for the memory amount. A user can only specify a memory value that's a multiple of `step`. For example, if `step: 512Mi`, the allowed values are 512Mi, 1Gi, 1.5Gi, 2Gi, and so on. A value of 700Mi is rejected because it isn't a multiple of 512Mi.

The choice of memory depends on the number of cores. If a user requests 5 cores, the Postgres object isn't created, since this configuration isn't provided by the administrator.

⚠ Within a single PostgresClass, the `cores.min`–`cores.max` ranges of different policies must not overlap.

For example, in the `production-v1` PostgresClass, the first policy can't cover 1–4 cores while the second covers 2–6 cores, since cores 2–4 would belong to both policies. Ranges can overlap across different PostgresClass resources, for example `production-v1` and `production-v2`.

The user can see the available options and pick a suitable one when creating a Postgres object.

Manage fault tolerance across availability zones

The administrator can manage the fault tolerance of users' PostgreSQL instances by setting their availability zone distribution topology in the `spec.topology` parameter section. This can be useful in production environments with data-center-level fault tolerance requirements.

Three topologies are available:

- `Ignored`: standard scheduling with no zone binding;
- `Zonal`: all instances are placed in one zone (minimal latency between replicas). Suits low-latency environments where losing a zone is acceptable;
- `TransZonal`: instances are distributed across different zones (one primary instance, one synchronous replica, one asynchronous replica). Protects against the loss of an entire zone but requires more resources.

The administrator can specify the allowed topology options (`allowedTopologies`), the default topology (`defaultTopology`), and the list of available zones (`allowedZones`).

Below is a fragment of a PostgresClass manifest showing the topology parameters. The complete example is provided in Example of creating a PostgresClass.

```
spec:
  topology:
    allowedTopologies:
      - Zonal
      - TransZonal
    defaultTopology: TransZonal
    allowedZones:
      - zone-a
      - zone-b
```

In this fragment, `Zonal` and `TransZonal` modes are allowed, with `TransZonal` applied by default. The `allowedZones` field lists placeholder zones — replace them with the actual zone names from your cloud provider or data center.

Users can choose the fault tolerance level when creating a Postgres object. If not specified explicitly, the default value applies (as set in the `defaultTopology` parameter).

Automatically validate PostgreSQL settings

To reject undesirable combinations of PostgreSQL parameters chosen by users, the administrator can define validation rules in Common Expression Language (CEL) in the `spec.validations` parameter section of PostgresClass. These rules are checked on the API server before Postgres objects are created.

These rules are useful for guarding against common configuration mistakes that can lead to performance issues. For example, too large a `shared_buffers` value can leave PostgreSQL with too little memory for other operations, and a large number of connections can significantly increase overall memory consumption.

The following variables are available in the rules:

- `configuration.maxConnections`;
- `configuration.workMem`;
- `configuration.sharedBuffers`;
- `configuration.walKeepSize`;
- `instance.memory.size`;
- `instance.cpu.cores`.

Below is a fragment of a PostgresClass manifest showing validation rules. The complete set of rules is provided in Example of creating a PostgresClass.

```
spec:
  validations:
    - message: "Shared buffers should not be more than 25% of RAM"
      rule: "configuration.sharedBuffers < instance.memory.size / 4"
```

In this fragment, the rule prevents allocating more than 25% of the requested memory to `shared_buffers`.

The user gets a validation error when trying to apply invalid parameters, which helps avoid configuration mistakes without manual review.

Set default values and allow overrides

To set default PostgreSQL parameters and define which of them users can change, the administrator can use the `spec.configuration` and `spec.overridableConfiguration` parameters.

If no values are specified in `configuration`, the module controller applies the following:

- `maxConnections`: `100`;
- `sharedBuffers`: 25% of `memory.size`;
- `workMem`: `(memory.size - sharedBuffers) * 4 / maxConnections`;
- `walKeepSize`: `512Mi`.

Below is a fragment of a PostgresClass manifest showing the default service settings. The complete example is provided in Example of creating a PostgresClass.

```
spec:
  configuration:
    maxConnections: 200
    sharedBuffers: 1Gi
  overridableConfiguration:
    - maxConnections
    - sharedBuffers
    - walKeepSize
```

In this example:

- `maxConnections` and `sharedBuffers` default to 200 and 1Gi, respectively.
- The user can override `maxConnections`, `sharedBuffers`, and `walKeepSize`.
- `workMem` isn’t included in `overridableConfiguration`, so the user can’t change it.

Automatic workMem calculation

To limit the amount of memory used for sort and hash operations within a single query, use the

workMem parameter.

In PostgreSQL, this limit applies to each operation in each active session. A complex query can run several such operations at once. With a large number of connections, total memory consumption can far exceed the configured value and lead to memory exhaustion. A value that's too small, on the other hand, forces PostgreSQL to use temporary files on disk and reduces performance.

If `workMem` isn't set explicitly, the `managed-postgres` module automatically calculates its value based on the allocated resources.

The module calculates `workMem` using the following formula:

```
workMem = (instance.memory.size - configuration.sharedBuffers) * 4 / configuration.maxConnections
```

For example, for Postgres with the following parameters:

- `instance.memory.size: 4Gi`
- `configuration.sharedBuffers: 1Gi`
- `configuration.maxConnections: 200`

The `workMem` calculation looks as follows:

```
workMem = (4Gi - 1Gi) * 4 / 200
workMem = 3Gi * 4 / 200
workMem = 12Gi / 200
workMem = 61.44Mi
```

The `* 4` multiplier is a safety margin for the case of several concurrent sort and hash operations within a single query.

The controller calculates `workMem` using the formula above, based on the instance's memory size, `sharedBuffers`, `maxConnections`, and the safety margin.

Bind to dedicated nodes

The standard Kubernetes mechanisms — `spec.nodeSelector`, `spec.tolerations`, and `spec.nodeAffinity` — let you specify which nodes PostgreSQL pods can be placed on.

Placement on dedicated nodes helps isolate PostgreSQL instances from user applications and makes disk and network resource usage more predictable.

Below is a fragment of a PostgresClass manifest showing `spec.nodeAffinity`, `spec.nodeSelector`, and `spec.tolerations`. The complete example is provided in Example of creating a PostgresClass.

```
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: "node.deckhouse.io/group"
              operator: "In"
              values:
                - "pg"
  nodeSelector:
    "node.deckhouse.io/group": "pg"
  tolerations:
    - key: primary-role
      operator: Equal
      value: pg
      effect: NoSchedule
```

In this example:

- Pods are placed only on nodes with the `node.deckhouse.io/group=pg` label.
- Nodes have the `primary-role=pg:NoSchedule` taint, which prevents scheduling pods without a matching toleration.
- `tolerations` allows PostgreSQL pods to be scheduled on such nodes.

Users don't need to worry about node selection. Pods are automatically placed on the prepared infrastructure according to the PostgresClass settings.

If no node meets the placement rules, users' Postgres objects will remain in the `Pending` state. To determine the cause, you can check the events in the namespace, the Postgres object information, and the PVC status:

```
d8 k describe pods -n my-postgres <POD_NAME>
d8 k get events -n my-postgres --sort-by=.lastTimestamp
d8 k get pvc -n my-postgres
```

Additional resources

- Module `cert-manager` documentation
- Module `managed-postgres` documentation
- Module `snapshot-controller` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Backup and restore

Manual cluster restore

Restoring a cluster with a single control plane node

To properly restore the cluster, follow these steps on the master node:

1. Prepare the `etcdutl` utility. Locate and copy the executable on the node:

   ```
   ETCD_PID=$(crictl inspect $(crictl ps --name etcd -q | head -1) | jq .info.pid)
   cp /proc/${ETCD_PID}/root/usr/bin/etcdutl /usr/local/bin/etcdutl
   chmod +x /usr/local/bin/etcdutl
   ```

   Check the version of `etcdutl`:

   ```
   etcdutl version
   ```

   Make sure the output of `etcdutl version` is displayed without errors.

   If `etcdutl` is not found, download the binary from the official etcd repository, choosing a version that matches your cluster's etcd version:

   ```
   wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
   tar -xzvf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdutl /usr/local/bin/etcdutl
   ```

2. Check the etcd version in the cluster (if the Kubernetes API is accessible):

   ```
   d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdutl version
   ```

   If the command executes successfully, it will display the current etcd version.

3. Stop etcd. Move the etcd manifest to prevent kubelet from launching the etcd pod:

   ```
   mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
   ```

4. Make sure the etcd pod is stopped:

   ```
   crictl ps | grep etcd
   ```

If the command returns nothing, the etcd pod has been successfully stopped.

5. Backup current etcd data. Create a backup copy of the `member` directory:

   ```
   cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
   ```

   This backup will allow you to roll back in case of issues.

6. Place the etcd snapshot file. Copy or move the `etcd-backup.snapshot` file to the current user's (root) home directory:

   ```
   cp /path/to/backup/etcd-backup.snapshot ~/etcd-backup.snapshot
   ```

   Ensure the file is readable:

   ```
   ls -la ~/etcd-backup.snapshot
   ```

7. Clean the etcd directory. Remove old data to prepare for restore:

   ```
   rm -rf /var/lib/etcd
   ```

   Verify that `/var/lib/etcd` is now empty or does not exist:

   ```
   ls -la /var/lib/etcd
   ```

8. Restore the etcd database from the snapshot using `etcdutl`:

   ```
   ETCDCTL_API=3 etcdutl snapshot restore ~/etcd-backup.snapshot  --data-dir=/var/lib/etcd
   ```

   After the command completes, check that files have appeared in `/var/lib/etcd/`, reflecting the restored state.

9. Start etcd. Move the manifest back so that kubelet relaunches the etcd pod:

   ```
   mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
   ```

10. Wait for the pod to be created and reach `Running` state. Make sure it is up and running:

    ```
    crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
    ```

    Pod startup may take some time. Once etcd is running, the cluster will be restored from the snapshot.

    Output example:

    ```
    CONTAINER        IMAGE            CREATED              STATE     NAME
    ATTEMPT     POD ID          POD
    4b11d6ea0338f    16d0a07aa1e26    About a minute ago   Running   etcd      0
    ee3c8c7d7bba6   etcd-gs-test
    ```

11. Restart the master node.

Restoring a multi-master cluster

To properly restore a multi-master cluster, follow these steps:

1. Enable High Availability (HA) mode. This is necessary to preserve at least one Prometheus replica and its PVC, since HA is disabled by default in single-master clusters.
2. Enable maintenance mode on the remaining master node to prevent `control-plane-manager` from creating new operations on it during the restore procedure (CPM will continue to remove components from the nodes being detached):

```
d8 k label cpn <REMAINING_MASTER_NAME> control-plane-manager.deckhouse.io/maintenance=""
```

3. Switch the cluster to single master mode:
   - In a cloud cluster, follow the instructions.
   - In a static cluster, remove any unnecessary master nodes from the `control-plane` role by following the instructions and then remove them from the cluster.
   - In a static cluster with the configured HA mode based on two master nodes and an arbiter node, remove the arbiter node and additional master nodes.
   - In a cloud cluster with the configured HA mode based on two master nodes and an arbiter node, use the instructions to remove the additional master nodes and the arbiter node.
4. Restore etcd from the backup on the only remaining master node. Follow the instructions for restoring a cluster with a single control-plane node.
5. Make sure the maintenance mode label is removed from the remaining master node (after restoring from the backup it is usually already gone, but it is better to verify explicitly):

   ```
   d8 k label cpn <REMAINING_MASTER_NAME> control-plane-manager.deckhouse.io/maintenance-
   ```
6. Once etcd is restored, remove the records of the previously deleted master nodes from the cluster using the following command (replace with the actual node name):

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

   > **Warning.** If the `d8 k` or `kubectl` commands are unavailable on the node, check the `/etc/kubernetes/kubernetes-api-proxy/upstreams.json` configuration file. It should only specify your current API server. If the configuration contains lines with IP addresses of old master nodes, remove them. Edit the configuration in a similar way on all other nodes.
7. Reboot the master node. Ensure that the other nodes transition to the `Ready` state.
8. Wait for Deckhouse to process all tasks in the queue:

   ```
   d8 system queue main
   ```
9. Switch the cluster back to multi-master mode. For cloud clusters, follow the instructions.

Once you go through these steps, the cluster will be successfully restored in the multi-master configuration.

Restoring individual objects

Restoring Kubernetes objects from an etcd backup

To restore individual cluster objects (e.g., specific Deployments, Secrets, or ConfigMaps) from an etcd snapshot, follow these steps:

1. Launch a temporary etcd instance. Create a separate etcd instance that runs independently from the main cluster.
2. Load data from backup copy into temporary etcd instance. Use the existing etcd snapshot file to populate the temporary instance with the necessary data.
3. Unload the manifests of the necessary objects in YAML format.
4. Restore cluster objects from uploaded YAML files.

Example of steps to restore objects from an etcd backup

In the example below, `etcd-backup.snapshot` is a etcd shapshot, `infra-production` is the namespace in which objects need to be restored.

- To decode objects from `etcd` you would need auger. It can be built from source on any machine that has Docker installed (it cannot be done on cluster nodes).

```
git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
cd auger
make release
build/auger -h
```

- Resulting executable `build/auger`, and also the `snapshot` from the backup copy of etcd must be uploaded on master-node, on which following actions would be performed.

Following actions are performed on a master node, to which `etcd snapshot` file and `auger` tool were copied:

1. Set the correct access permissions for the backup file:

   ```
   chmod 644 etcd-backup.snapshot
   ```

2. Set full path for snapshot file and for the tool into environmental variables:

   ```
   SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot
   AUGER_BIN=/root/auger
   chmod +x $AUGER_BIN
   ```

3. Run a Pod with temporary instance of `etcd`.
   - Create Pod manifest. It should schedule on current master node by `$HOSTNAME` variable, and mounts snapshot file by `$SNAPSHOT` variable, which it then restores in temporary `etcd` instance:

```
cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
  - operator: Exists
  initContainers:
  - command:
    - etcdutl
    - snapshot
    - restore
    - "/tmp/etcd-snapshot"
    - --data-dir=/default.etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-snapshot-restore
    # Uncomment the fragment below to set limits for the container if the
node does not have enough resources to run it.
    # resources:
    #   requests:
    #     ephemeral-storage: "200Mi"
    #   limits:
    #     ephemeral-storage: "500Mi"
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
    - name: etcd-snapshot
      mountPath: /tmp/etcd-snapshot
      readOnly: true
  containers:
  - command:
    - etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-temp
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
  volumes:
  - name: etcddir
    emptyDir: {}
    # Use the snippet below instead of emptyDir: {} to set limits for the
container if the node's resources are insufficient to run it.
    # emptyDir:
    #  sizeLimit: 500Mi
  - name: etcd-snapshot
    hostPath:
      path: $SNAPSHOT
      type: File
EOF
```

- Create Pod from the resulting manifest:

```
d8 k create -f etcd.pod.yaml
```

4. Set environment variables. In this example:
   - `infra-production` — namespace which we will search resources in.
   - `/root/etcd-restore/output` — path for outputting recovered resource manifests.
   - `/root/auger` — path to `auger` executable.

   ```
   FILTER=infra-production
   BACKUP_OUTPUT_DIR=/root/etcd-restore/output
   mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
   ```

5. Commands below will filter needed resources by `$FILTER` and output them into `$BACKUP_OUTPUT_DIR` directory:

   ```
   files=($(kubectl -n default exec etcdrestore -c etcd-temp -- etcdctl  --
   endpoints=localhost:2379 get / --prefix --keys-only | grep "$FILTER"))
   for file in "${files[@]}"
   do
     OBJECT=$(kubectl -n default exec etcdrestore -c etcd-temp -- etcdctl  --
   endpoints=localhost:2379 get "$file" --print-value-only | $AUGER_BIN decode)
    FILENAME=$(echo $file | sed -e "s#/registry/##g;s#/#_#g")
    echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
    echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
   done
   ```

6. Restore objects from exported YAML files.
7. Delete the Pod with a temporary instance of etcd:

   ```
   d8 k -n default delete pod etcdrestore
   ```

Restoring cluster objects from exported YAML files

To restore objects from exported YAML files, follow these steps:

1. Prepare the YAML files for restoration. Before applying them to the cluster, remove technical fields that may be outdated or interfere with the recovery process:
   - `creationTimestamp`
   - `UID`
   - `status`

   You can edit these fields manually or use YAML/JSON processing tools such as `yq` or `jq`.
2. Create the objects in the cluster. To restore individual resources, run:

   ```
   d8 k create -f <PATH_TO_FILE>.json
   ```

   You can specify either a single file or a directory path.
3. To restore multiple objects at once, use the `find` command:

   ```
   find $BACKUP_OUTPUT_DIR -type f -name "*.yaml" -exec d8 k create -f {} \;
   ```

   This will locate all `.yaml` files within the specified `$BACKUP_OUTPUT_DIR` and apply them sequentially using `d8 k create`.

After completing these steps, the selected objects will be recreated in the cluster based on the definitions in the YAML files.

Restoring the licensing cluster key

Deckhouse Platform licensing keeps the cluster key and the registration counter in two objects:

- Secret `d8-system/cluster-key` holds the cluster key.
- ConfigMap `d8-system/license-registration` holds the counter of cluster data files. The

customer portal accepts a cluster data file only if its number is greater than the numbers of the files uploaded before.

A license key is issued for the cluster UUID, the one in the ConfigMap `kube-system/d8-cluster-uuid`, and can also be bound to the cluster key.

If the cluster has these objects, include both of them in its backups. An etcd backup contains them. To save them separately, run:

```
d8 k -n d8-system get configmap license-registration -o yaml > license-registration.yaml
d8 k -n d8-system get secret cluster-key -o yaml > cluster-key.yaml
```

Store `cluster-key.yaml` as securely as the other secrets of the cluster.

If the cluster key is damaged, or Deckhouse Platform has created a new one because the Secret was lost or the cluster was rebuilt from a backup, the license keys bound to the old cluster key stop applying. If the cluster still has the UUID the license keys were issued for, restore the cluster key from a backup. A cluster with another UUID needs license keys of its own whatever its cluster key: get them in the customer portal.

Only Deckhouse Platform and `system:sudouser` may change the two objects, so a backup tool cannot restore them on behalf of its own account. Restore them manually:

1. Take the manifests of both objects from the same backup, the most recent one, and prepare them as described in Restoring cluster objects from exported YAML files. Also remove the `resourceVersion` and `managedFields` fields.
2. Apply the manifests on behalf of `system:sudouser`. The command replaces the objects that Deckhouse Platform has created in their place:

   ```
   d8 k --as=system:sudouser apply --server-side --force-conflicts -f license-registration.yaml -f cluster-key.yaml
   ```

Deckhouse Platform picks up the restored key within an hour. After that, the license keys issued for the cluster apply again: both the ones restored from the backup and the ones uploaded again in the Deckhouse web UI section “Licensing”. To list the records that are still rejected as issued for another cluster key, run:

```
d8 k get clusterlicenses -o jsonpath='{range .items[*]}{range .status.records[?(@.reason=="ClusterKeyMismatch")]}{.id}: {.message}{"\n"}{end}{range .status.records[?(@.reason=="ClusterKeyMissing")]}{.id}: {.message}{"\n"}{end}{end}'
```

Once the key is picked up, the output is empty. A record rejected with the reason `ClusterMismatch` was issued for another cluster UUID, and restoring the key does not change it.

Restoring control plane component manifests from a control-plane-manager backup

Before applying any change to a control plane component (`etcd`, `kube-apiserver`, `kube-controller-manager`, or `kube-scheduler`), the `control-plane-manager` module automatically backs up the files it is about to overwrite on that node. Backups are stored per component and per operation in:

```
/etc/kubernetes/deckhouse/backup/<Component>/<operation-name>/
```

The `control-plane-manager` module keeps the 7 most recent operations per component. Older backups are removed automatically. The contents of a backup directory depend on the component. For example, for `kube-apiserver`, its static pod manifest, TLS certificates, and the `admin.conf` file are saved:

```
/etc/kubernetes/deckhouse/backup/KubeAPIServer/kubeapiserver-ef53e5-n2snk/
├── manifests/kube-apiserver.yaml
├── pki/apiserver.crt
├── pki/apiserver.key
└── admin.conf
```

Next to the backup in the `/etc/kubernetes/deckhouse/diffs/<component>/<operation-name>/` directory, the module keeps a unified diff file containing the differences of the changes made. For example:

```
/etc/kubernetes/deckhouse/diffs/KubeAPIServer/kubeapiserver-ef53e5-n2snk/
├── manifests/kube-apiserver.yaml.diff
└── extra-files/...
```

You can use these backups to revert a control plane component to its previous state — for example, after a change applied by `control-plane-manager` (such as one triggered by editing a ModuleConfig) broke the component.

To restore a control plane component file from a `control-plane-manager` backup, do the following:

1. On the target master node, list the available backups for the component and pick the operation to restore from:

   ```
   ls -lt /etc/kubernetes/deckhouse/backup/<COMPONENT>/
   ```

   Optionally, inspect the corresponding diff file to see what changed in that operation:

   ```
   cat /etc/kubernetes/deckhouse/diffs/<COMPONENT>/<OPERATION_NAME>/manifests/*.diff
   ```

2. Enable maintenance mode on the node's ControlPlaneNode resource. This prevents `control-plane-manager` from starting new operations on the node while you are restoring files manually. An operation already in progress is not affected.

   ```
   d8 k label cpn <NODE_NAME> control-plane-manager.deckhouse.io/maintenance=""
   ```

3. Copy the required file(s) from the backup directory back to their original location, preserving their relative path under `/etc/kubernetes`. For example, to restore the `kube-apiserver` manifest:

   ```
   cp /etc/kubernetes/deckhouse/backup/KubeAPIServer/<OPERATION_NAME>/manifests/kube-apiserver.yaml
   /etc/kubernetes/manifests/kube-apiserver.yaml
   ```

   Restoring a static pod manifest is enough to trigger kubelet to restart the pod automatically, and no manual restart is required.

4. If you restored certificates, keys, `admin.conf`, and similar files under `pki/` but didn't change the manifest itself, restart the corresponding component's pod manually. Kubelet only tracks changes in the manifest file and doesn't restart the container when mounted files are changed.

   Deleting the pod is enough for kubelet to recreate it from the unchanged static manifest:

   ```
   d8 k -n kube-system delete po <COMPONENT_POD_NAME>
   ```

   Here `<COMPONENT_POD_NAME>` is the static pod's name, for example, `kube-apiserver-<NODE_NAME>`.

   If the API server itself is unreachable (for example, if you are restoring a certificate of a single control plane node, making the API server unreachable), restart the container directly on the node instead:

   ```
   crictl stopp $(crictl pods --name=<COMPONENT_NAME> -q)
   ```

5. Make sure the component came back up healthy:

   ```
   d8 k -n kube-system wait pod \
     --timeout=10m \
     --for=condition=ContainersReady \
     -l app=d8-control-plane-manager \
     --field-selector spec.nodeName=<NODE_NAME>
   ```

6. Disable maintenance mode:

   ```
   d8 k label cpn <NODE_NAME> control-plane-manager.deckhouse.io/maintenance-
   ```

⚠ Restoring files from a backup only reverts their on-disk state. If the change you're reverting is still requested by the current cluster configuration (for example, set in a ModuleConfig), `control-plane-manager` will detect the drift and create a new operation to reapply it as soon as maintenance mode is disabled. If you want to keep the restored state, modify the initial cluster configuration first.

Restoring objects after changing the master node IP address

⚠ This section describes a scenario where only the IP address of the master node has changed, and all other objects in the etcd backup (such as CA certificates) remain valid. It assumes the restoration is performed in a single-master-node cluster.

To restore etcd objects after changing the master node's IP address, follow these steps:

1. Restore etcd from the backup. Use the standard etcd restore procedure with a snapshot. Make sure not to change any parameters during restoration other than the etcd data itself.
2. Update the IP address in static configuration files:
   - Check the Kubernetes component manifest files located in `/etc/kubernetes/manifests/`.
   - Review kubelet's system configuration files (typically found in `/etc/systemd/system/kubelet.service.d/` or similar directories).
   - Update the IP address in any other configurations that reference the old address, if necessary.
3. Back up the current certificates:

   ```
   cp -r /etc/kubernetes/pki ./pki-backup
   ```

4. Renew control-plane certificates with the new IP address added to SANs:

   ```
   d8 tools pki certs renew all --san <NEW_IP>
   ```

5. Restart all services so that components load the updated certificates and configurations.
6. Wait for kubelet to regenerate its own certificate.

These actions can be performed either automatically using a script, or manually by running the required commands step-by-step.

Automated object extraction when changing IP address

To simplify cluster recovery after the master node's IP address changes, use the script provided below. Before running the script:

1. Specify the correct paths and IP addresses:
   - `ETCD_SNAPSHOT_PATH`: The path to the etcd snapshot backup.
   - `OLD_IP`: The old master node IP address used when the backup was created.
   - `NEW_IP`: The new IP address of the master node.
2. Download `etcdutl` if it is not installed.
3. After running the script, wait for the kubelet to regenerate its certificate with the new IP address. You can verify this in the `/var/lib/kubelet/pki/` directory, where a new certificate should appear.

Object extraction script

```
ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Path to the etcd snapshot.
OLD_IP=10.242.32.34                         # Old master node IP address.
NEW_IP=10.242.32.21                         # New master node IP address.
NODE_NAME=master-0                          # Must match --name in the etcd static pod
manifest.

mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
mkdir ./etcd_old
mv /var/lib/etcd ./etcd_old
ETCD_PID=$(crictl inspect $(crictl ps --name etcd -q | head -1) | jq .info.pid)
ETCDUTL_PATH=/proc/${ETCD_PID}/root/usr/bin/etcdutl

$ETCDUTL_PATH snapshot restore "$ETCD_SNAPSHOT_PATH" \
  --data-dir=/var/lib/etcd \
  --name="$NODE_NAME" \
  --initial-advertise-peer-urls="https://$NEW_IP:2380" \
  --initial-cluster="$NODE_NAME=https://$NEW_IP:2380"

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
sed -i "s/$OLD_IP/$NEW_IP/g" ~/etcd.yaml

cp -r /etc/kubernetes/pki ./pki-backup

d8 tools pki certs renew all --san $NEW_IP

mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml

crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs crictl
stop
crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

systemctl daemon-reload
systemctl restart kubelet.service
```

Manual object restore after changing the IP address

If you prefer to manually make changes during cluster recovery after the master node’s IP address has changed, follow these steps:

1. Restore etcd from the backup:
   - Move the etcd manifest so that kubelet stops the corresponding pod:

     ```
     mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
     ```

   - Create a directory to temporarily store the previous etcd data:

     ```
     mkdir ./etcd_old
     mv /var/lib/etcd ./etcd_old
     ```

   - Find or download the `etcdutl` utility if it’s not available, and perform the snapshot restore:

```
ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Path to the etcd snapshot.
NEW_IP=10.242.32.21                         # New master node IP address.
NODE_NAME=master-0                          # Must match --name in the etcd static
pod manifest.
ETCD_PID=$(crictl inspect $(crictl ps --name etcd -q | head -1) | jq .info.pid)
ETCDUTL_PATH=/proc/${ETCD_PID}/root/usr/bin/etcdutl

$ETCDUTL_PATH snapshot restore "$ETCD_SNAPSHOT_PATH" \
  --data-dir=/var/lib/etcd \
  --name="$NODE_NAME" \
  --initial-advertise-peer-urls="https://$NEW_IP:2380" \
  --initial-cluster="$NODE_NAME=https://$NEW_IP:2380"
```

2. Update the IP address in static configuration files. If the old IP address is used in manifests or kubelet services, replace it with the new one:

```
OLD_IP=10.242.32.34                     # Old master node IP address.
NEW_IP=10.242.32.21                     # New master node IP address.

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {}
';'
find /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
sed -i "s/$OLD_IP/$NEW_IP/g" ~/etcd.yaml
```

3. Back up the current certificates:

```
cp -r /etc/kubernetes/pki ./pki-backup
```

4. Renew control-plane certificates with the new IP address added to SANs:

```
d8 tools pki certs renew all --san <NEW_IP>
```

5. Restore the etcd manifest (so kubelet starts the etcd pod again) and restart all services so that components load the updated certificates and configurations. To immediately stop active containers, run:

```
mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml

crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl
stop
crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs
crictl stop
crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

systemctl daemon-reload
systemctl restart kubelet.service
```

   Verify etcd is running by checking the pod list using `crictl ps | grep etcd` or reviewing the kubelet logs.

6. Wait for kubelet to regenerate its own certificate. Kubelet will automatically generate a new certificate with the updated IP address:
   - Check the `/var/lib/kubelet/pki/` directory.
   - Ensure the new certificate is present and valid.

Once these steps are completed, the cluster will be successfully restored and fully functional with the new master node IP address.

Creating backups with Deckhouse CLI

Deckhouse CLI (`d8`) provides the `backup` command for creating backups of various cluster components:

- `etcd`: Snapshot of the Deckhouse key-value data store.

- `cluster-config`: Archive containing key configuration objects of the cluster.
- `loki`: Export of logs from the built-in Loki API.

Backing up etcd

An etcd snapshot allows you to preserve the current state of the cluster at the key-value storage level. This is a full dump that can be used for recovery.

To create a snapshot, run the following command:

```
d8 backup etcd <path-to-snapshot> [flags]
```

Flags:

- `-p`, `--etcd-pod string`: Name of the etcd pod to snapshot.
- `-h`, `--help`: Show help for the etcd command.
- `--verbose`: Enable verbose output for detailed logging.

Example:

```
d8 backup etcd etcd-backup.snapshot
```

Example output:

```
2025/04/22 08:38:58 Trying to snapshot etcd-sandbox-master-0
2025/04/22 08:39:01 Snapshot successfully taken from etcd-sandbox-master-0
```

Automatic etcd backup

Deckhouse automatically performs a daily etcd backup using a CronJob that runs inside the `d8-etcd-backup` pod in the `kube-system` namespace. The job creates a snapshot of the database, compresses it, and saves the archive locally on the node at `/var/lib/etcd/`:

```
etcdctl snapshot save etcd-backup.snapshot
tar -czvf etcd-backup.tar.gz etcd-backup.snapshot
mv etcd-backup.tar.gz /var/lib/etcd/etcd-backup.tar.gz
```

To configure automatic etcd backups, use the `control-plane-manager` module. The required parameters are specified in its configuration:

Parameter	Description
`etcd.backup.enabled`	Enables daily etcd backup.
`etcd.backup.cronSchedule`	Cron-formatted schedule for running the backup. Local time of `kube-controller-manager` is used.
`etcd.backup.hostPath`	Path on master nodes where etcd backup archives will be stored.

Example configuration fragment:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    etcd:
      backup:
        cronSchedule: 0 14 * * *
        enabled: true
```

Backups on different master nodes run with a 15-minute offset from each other, so the archives on different nodes can contain snapshots taken at different times. If a backup job fails on a node, its archive can be even older. To restore, use the snapshot with the highest revision. To find out the revision, unpack the archive and run:

```
tar -xzf etcd-backup.tar.gz
etcdutl snapshot status etcd-backup.snapshot -w table
```

Cluster configuration backup

The `d8 backup cluster-config` command creates an archive containing a set of key resources related to the cluster configuration. This is not a full backup of all objects, but a specific whitelist.

To create the backup, run the following command:

```
d8 backup cluster-config <path-to-backup-file>
```

Example:

```
d8 backup cluster-config /backup/cluster-config-2025-04-21.tar
```

The archive includes only those objects that meet the following criteria:

- CustomResource objects whose CRDs are annotated with:

  ```
  backup.deckhouse.io/cluster-config=true
  ```

- StorageClasses with the label:

  ```
  heritage=deckhouse
  ```

- Secrets and ConfigMaps from namespaces starting with `d8-` or `kube-`, if they are explicitly listed in the whitelist file.
- Cluster-level Roles and Bindings (ClusterRole and ClusterRoleBinding), if they are not labeled with:

  ```
  heritage=deckhouse
  ```

> **i** The backup includes only CR objects, but not the CRD definitions themselves. To fully restore the cluster, the corresponding CRDs must already be present (e.g., installed by Deckhouse modules).

Example whitelist content:

Namespace	Object	Name
`d8-system`	Secret	`d8-cluster-terraform-state`
		`$regexp:^d8-node-terraform-state-(.*)$`
		`deckhouse-registry`
	ConfigMap	`d8-deckhouse-version-info`
`kube-system`	ConfigMap	`d8-cluster-is-bootstraped`
		`d8-cluster-uuid`
		`extension-apiserver-authentication`
	Secret	`d8-cloud-provider-discovery-data`
		`d8-cluster-configuration`
		`d8-cni-configuration`

Namespace	Object	Name
		`d8-control-plane-manager-config`
		`d8-node-manager-cloud-provider`
		`d8-pki`
		`d8-provider-cluster-configuration`
		`d8-static-cluster-configuration`
		`d8-secret-encryption-key`
`d8-cert-manager`	Secret	`cert-manager-letsencrypt-private-key`
		`selfsigned-ca-key-pair`

Exporting logs from Loki

The `d8 backup loki` command is intended for exporting logs from the built-in Loki. This is not a full backup, but rather a diagnostic export: the resulting data cannot be restored back into Loki.

To perform the export, `d8` accesses the Loki API using the `loki` ServiceAccount in the `d8-monitoring` namespace, authenticated via a token stored in a Kubernetes secret.

The `loki` ServiceAccount is automatically created starting from Deckhouse v1.69.0. However, to use the `d8 backup loki` command, you must manually create the token secret and assign the necessary Role and RoleBinding if they are not already present.

Apply the manifests below before running `d8 backup loki` to ensure the command can properly authenticate and access the Loki API.

Example manifests:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: loki-api-token
  namespace: d8-monitoring
  annotations:
    kubernetes.io/service-account.name: loki
type: kubernetes.io/service-account-token
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
rules:
  - apiGroups: ["apps"]
    resources:
      - "statefulsets/http"
    resourceNames: ["loki"]
    verbs: ["create", "get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: access-to-loki-from-d8
subjects:
  - kind: ServiceAccount
    name: loki
    namespace: d8-monitoring
```

To create a log backup, run the following command:

```
d8 backup loki [flags]
```

Example:

```
d8 backup loki --days 1 > ./loki.log
```

Flags:

- `--start`, `--end`: Time range boundaries in the format “YYYY-MM-DD HH:MM:SS”.
- `--days`: The time window size for log export (default is 5 days).
- `--limit`: The maximum number of log lines per request (default is 5000).

You can list all available flags using the following command:

```
d8 backup loki --help
```

Additional resources

- Module `control-plane-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Automatic assignment of namespace labels and annotations

You can automate the assignment of labels and annotations to namespaces in a Deckhouse cluster based on predefined patterns. This can be useful, for example, to automatically include new namespaces in monitoring without having to manually edit each one.

How it works

- All namespaces whose names match the patterns in `includeNames` and do not match patterns in `excludeNames` automatically receive the specified labels and annotations.
- When the configuration is changed, labels and annotations of existing namespaces are updated accordingly.
- Newly created namespaces that match the defined patterns also receive the specified labels and annotations automatically.

Configuring automatic label and annotation assignment

Enable the `namespace-configurator` module:

```
d8 system module enable namespace-configurator
```

Configure automatic label and annotation assignment in ModuleConfig `namespace-configurator`:

1. Specify the annotations and labels to be applied to namespaces in the fields `settings.configurations.annotations` and `settings.configurations.labels`, respectively;
2. Define the matching rules for namespace names:
   - In `includeNames`, list regular expressions for names you want to match.
   - In `excludeNames`, list names that should be excluded.

Example configuration

In the following configuration example, automatic addition of the label `extended-monitoring.deckhouse.io/enabled=true` and the annotation `foo=bar` is configured for all namespaces whose names start with `prod-` or `infra-`, except for `infra-test`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
    - annotations:
        foo: bar
      labels:
        extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
      - "^prod"
      - "^infra"
      excludeNames:
      - "infra-test"
```

Additional resources

- Module `namespace-configurator` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Application delivery with Argo CD

This section describes application delivery with Argo CD in Deckhouse Platform (DP).

Argo CD is an Open Source tool for continuous delivery of applications to Kubernetes that implements the GitOps approach. A Git repository is the source of truth for describing applications, their configuration, and target environments.

Argo CD tracks changes, shows differences between the desired and actual resource state, and when needed brings the cluster to the target state automatically or manually.

Main Argo CD capabilities:

- automatic and manual application deployment in Kubernetes;
- visualization of differences between the desired and actual state;
- tracking application configuration in Git;
- working with multiple Kubernetes clusters;
- access control through its own role model;
- rollback to a previously recorded configuration and action audit;
- working through the web interface, CLI, and API, including integration with Git webhooks and external automation systems.

In DP, Argo CD is run with the operator-argo module, based on the Argo CD Operator project. The module lets you declaratively deploy and maintain one or more Argo CD instances in a DP cluster without manually installing and maintaining components.

Additionally, the module provides:

- single sign-on (SSO) support;
- integration with monitoring in DP;
- high availability mode;
- support for automatic image updates via Argo CD Image Updater;
- access control for cluster resources.

A typical administrator setup scenario:

1. Enable the `operator-argo` module and create an ArgoCD object.
2. Configure user authentication.
3. Define the role-based access model.
4. Give users access to the Argo CD instance for application delivery (for more details, see the Usage section).

The design of the `operator-argo` module is described in the Architecture section. Module parameters and custom resources are documented in the `operator-argo` module documentation.

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Running Argo CD

This section describes the steps required to run an Argo CD instance in a DP cluster:

- Preparing to run Argo CD
- Deploying one or multiple Argo CD instances
- Additional settings for Argo CD

Preparing to run Argo CD

Before creating Argo CD instances, complete the following steps:

1. Enable the `operator-argo` module.
2. Wait until the module switches to the `Ready` state.

   You can check the module state in the DP web interface or with the following command:

   ```
   d8 k get module operator-argo -W
   ```

Detailed information about module settings is available in the `operator-argo` module documentation.

After you enable the `operator-argo` module, Argo CD custom resources become available in the DP cluster.

To run an Argo CD instance, create an ArgoCD object. Working with custom resources that belong to an Argo CD instance is described in the Usage section.

List of Argo CD custom resources created by the `operator-argo` module...

CRD	Scope	Purpose
`argocds.argoproj.io`	**`operator-argo` module**	A custom resource used by the `operator-argo` module to configure and maintain an Argo CD instance in the target environment. Installation parameters, component set, integrations, access policies, and other instance settings are defined through the ArgoCD object.
`appprojects.argoproj.io`	**Argo CD instance**	Used for logical segmentation of applications and for defining policies: allowed Git repositories, target clusters and namespaces, as well as access rules and restrictions on resource usage.
`applications.argoproj.io`	**Argo CD instance**	The main Argo CD application CRD for describing an application that must be synchronized from a declarative source (Git, Helm, Kustomize, and others) into Kubernetes. Defines the source, target environment, and synchronization parameters.
`applicationsets.argoproj.io`	**Argo CD instance**	Used for automated creation of a set of Application objects from a template. Suitable for mass application management scenarios across multiple clusters, environments, directories, teams, or repository branches.

CRD	Scope	Purpose
`argocdexports.argoproj.io`	**Argo CD instance**	Used for declarative export of data related to an Argo CD instance to external systems or related DP components. Typically applied in integration scenarios where configuration, status, or access information must be published in a formalized way.
`namespacemanagements.argoproj.io`	**Argo CD instance**	Used to manage the lifecycle of namespaces within GitOps processes. Can automate creation, configuration, and maintenance of namespaces into which applications are then deployed.
`notificationsconfigurations.argoproj.io`	**Argo CD instance**	Used to configure the Argo CD notification mechanism. Lets you declaratively describe delivery channels and rules for sending events related to synchronization, deployment errors, application status changes, and other operational events.
`imageupdaters.argocd-image-updater.argoproj.io`	**Argo CD instance**	Used to automatically track new container image versions and update application parameters according to the defined versioning and publishing strategy.

Deploying an Argo CD instance

Parameters available for configuring an Argo CD instance are listed in the `operator-argo` module documentation.

To deploy an Argo CD instance in the `argocd` namespace and publish the Argo CD web interface through Ingress, use the following example (provide your own values for `<ARGOCD_DOMAIN>` and `<TLS_SECRET_NAME>`):

```
apiVersion: v1
kind: Namespace
metadata:
  name: argocd
---
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: <TLS_SECRET_NAME>
    insecure: true
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: argocd-ingress
  namespace: argocd
spec:
  dnsNames:
    - <ARGOCD_DOMAIN>
  issuerRef:
    kind: ClusterIssuer
    name: letsencrypt
  secretName: <TLS_SECRET_NAME>
```

⚠ The `spec.server.insecure: true` parameter in the example above disables internal TLS on the Argo CD API server. This helps avoid redirect loops when publishing through Ingress.

After you create the ArgoCD object, Argo CD components are started in the `argocd` namespace:

```
d8 k -n argocd get pods
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0       1/1     Running   0          35m
argocd-dex-server-759fff8444-zglp4    1/1     Running   0          2d23h
argocd-redis-568f5b889c-jg5dr         1/1     Running   0          4d
argocd-repo-server-78d9d6bcc6-9rwcm   1/1     Running   0          3d22h
argocd-server-76597597f9-kfqdl        1/1     Running   0          35m
```

When all components switch to the `Running` status, the web interface of the Argo CD instance becomes available at the address specified in the `spec.server.ingress.tls.hosts` parameter (in the example — `https://<ARGOCD_DOMAIN>`). Authentication setup and credentials retrieval are described in the Configuring authentication and authorization section.

Application delivery with Argo CD is described in the Usage section.

i You can work with Argo CD not only through the web interface and custom resources, but also with the `argocd` CLI utility. You can download the binary from the “Documentation” section of the Argo CD web interface. To get help for the CLI utility, run `argocd --help`.

Deploying multiple Argo CD instances

If the cluster needs multiple Argo CD instances, create a separate namespace (or DP project) and a separate ArgoCD object for each of them.

For example, create:

- a separate instance for the production environment;
- a separate instance for test environments;
- a separate instance for a specific team or project.

With this approach, each Argo CD instance is managed independently and has its own configuration.

⚠ Creating more than one ArgoCD object in a single namespace is not supported.

Advanced settings

Enabling high availability mode

To enable high availability mode, set the `spec.ha.enabled: true` parameter in the ArgoCD object.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  ha:
    enabled: true
    resources:
      requests:
        cpu: "500m"
        memory: "512Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
```

⚠ High availability mode provides high availability only for the `Redis` state store using `HAProxy`. It does not automatically make all Argo CD components highly available.

i

Running Argo CD in high availability mode requires at least three cluster nodes because of `pod anti-affinity` rules. Clusters that use IPv6 only are not supported.

When high availability mode is enabled, changes in `.spec.redis.resources` are not applied. Configure resource limits and requests for `Redis` through the `.spec.ha.resources` parameter.

Granting access to cluster resources

By default, an Argo CD instance receives privileges only for the namespace where it runs and for namespaces labeled with `argocd.argoproj.io/managed-by`. The label value must match the name of the namespace where the Argo CD instance runs.

To allow creation of cluster-wide resources, specify the namespace of the target Argo CD instance in the `clusterConfigNamespaces` parameter of the `operator-argo` module settings.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-argo
spec:
  enabled: true
  settings:
    clusterConfigNamespaces: argocd
  version: 1
```

If there are multiple Argo CD instances, list them in the `clusterConfigNamespaces` parameter as a comma-separated list.

After you change the `ModuleConfig`, the following ClusterRole and ClusterRoleBinding objects are created automatically:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - impersonate
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd-application-controller
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-application-controller
subjects:
- kind: ServiceAccount
  name: argocd-argocd-application-controller
  namespace: argocd
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-server
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - get
  - delete
  - patch
- apiGroups:
  - argoproj.io
  resources:
  - applications
  - applicationsets
  verbs:
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - list
- apiGroups:
  - batch
  resources:
  - jobs
  - cronjobs
  - cronjobs/finalizers
  verbs:
  - create
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd-server
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-server
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-server
subjects:
- kind: ServiceAccount
```

```
  name: argocd-argocd-server
  namespace: argocd
```

> **i** If the privileges described in the default cluster roles are excessive, set `spec.defaultClusterScopedRoleDisabled: true` in the ArgoCD object settings. In this case, cluster roles are not created automatically, and you can define the required privilege level for the ServiceAccount used by the Argo CD instance yourself.

> **i** If needed, you can override the access level to cluster resources at the AppProject object level using the `spec.clusterResourceBlacklist` and `spec.clusterResourceWhitelist` parameters.

Using a custom cluster domain

If the cluster uses a domain other than `cluster.local`, specify it in the `spec.clusterDomain` parameter of the ArgoCD object.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  clusterDomain: prod.local
```

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring authentication and authorization

Argo CD supports local authentication and is also integrated with the Deckhouse Platform identity and access subsystem. You can learn more about configuring authentication and authorization in the DP documentation.

> **⚠** If no additional settings are defined in the ArgoCD object, the local `admin` account with the `admin` role is active by default.

You can learn about managing user and group permissions in the Configuring role-based access control section.

Local authentication

When an ArgoCD object is created, the `admin` user with the `admin` role is created automatically. The `admin` user password is generated automatically. To retrieve it, run:

```
d8 k -n argocd get secret argocd-cluster -o jsonpath='{.data.admin\.password}' | base64 -d
```

⚠ We recommend changing the automatically generated `admin` user password using Argo CD CLI command:

```
argocd login <ARGOCD_DOMAIN>:443 --username admin --password <ADMIN_PASSWORD>
argocd account update-password \
  --account admin \
  --current-password <ADMIN_PASSWORD> \
  --new-password <NEW_PASSWORD>
```

Creating additional local users

When creating a local user, you can define whether the user has access to the Argo CD web interface (the `login` attribute) and/or to the Argo CD API (the `apiKey` attribute).

To create local users, set the `spec.localUsers` parameter of ArgoCD, for example:

```
d8 k create -f - <<EOF
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  localUsers:
    # The "deploy" user with rights to sign in to the web interface and generate
tokens.
    - name: deploy
      apiKey: true
      login: true
      enabled: true
    # The "ci-bot" user for automation only, without web interface access.
    - name: ci-bot
      apiKey: true
      login: false
      enabled: true
EOF
```

After creating the user and applying the ArgoCD object manifest, generate a password for web interface access if the user has been granted the corresponding rights.

To set a password through a Secret, first calculate its bcrypt hash, then encode the result in Base64:

```
ARGOCD_USER=<USER_NAME>
ARGOCD_PASS=$(echo -n '<USER_PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n'
| base64 -w0)
d8 k -n argocd patch secret argocd-secret -p "{\"data\":
{\"accounts.$ARGOCD_USER.password\":\"$ARGOCD_PASS\"}}"
```

i You can set or change a local user password with the `argocd` CLI utility:

```
argocd login <ARGOCD_DOMAIN>:443 --username admin --password <ADMIN_PASSWORD>
argocd account update-password \
  --account <ACCOUNT> \
  --current-password <ADMIN_PASSWORD> \
  --new-password <NEW_PASSWORD>
```

Creating a user token

To work with the API, the user must have the corresponding rights (the `apiKey` parameter; see the

example in the section above). To issue a token, use the Argo CD web interface or the `argocd` CLI utility.

To issue a token through the web interface, go to “Settings” → “Accounts”, select the required user, and click “Generate New” in the “Tokens” section.

To issue a token with the `argocd` CLI utility, run the following commands (provide the required values):

```
argocd login <ARGOCD_DOMAIN>:443 --username admin
argocd account generate-token --account <ACCOUNT>
```

> **i** If the user does not have web interface access rights (`login`), another user with administrator rights can issue a token for them.

Disabling local authentication

To disable local authentication, disable the `admin` user by setting the `spec.disableAdmin` parameter of ArgoCD to `true`. Also remove all additional local users if they were created earlier (for details, see Creating additional local users).

After disabling local authentication, review the access rules in the Configuring role-based access control section.

SSO authentication

Before configuring the ArgoCD object, create a DexClient object. It is used as an OAuth2 client and is required for integration with Deckhouse Platform:

```
d8 k create -f -<<EOF
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: argocd
  namespace: argocd
spec:
  redirectURIs:
    - https://<ARGOCD_DOMAIN>/api/dex/callback
    - https://<ARGOCD_DOMAIN>/api/dex/callback-reserve
EOF
```

`<ARGOCD_DOMAIN>` is the fully qualified domain name (FQDN) set in the `.spec.server.host` section of ArgoCD.

Wait until Deckhouse Platform creates a Secret with the client secret key:

```
d8 k -n argocd get secret/dex-client-argocd
```

Configure the ArgoCD object to use SSO in Deckhouse Platform:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    scopes: "[groups, email]"
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
            config:
              issuer: "https://dex.<CLUSTER_DOMAIN>/"
              clientID: "dex-client-argocd@argocd"
              clientSecret: "$dex-client-argocd:clientSecret"
              insecureEnableGroups: true
              scopes:
                - profile
                - email
                - openid
                - groups
    provider: dex
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      ingressClassName: <INGRESS_CLASS_NAME>
      tls:
        - hosts:
           - <ARGOCD_DOMAIN>
          secretName: argocd-ingress-tls
    insecure: true
```

Restart the Argo CD server:

```
d8 k -n argocd rollout restart deploy/argocd-server
```

⚠ If you do not restart the Argo CD server, the sign-in attempt fails, and an error message appears in the Argo CD server log (issue argoproj/argo-cd#13526).

Example error message...

```
time="2024-10-16T14:12:59Z" level=warning msg="Failed to verify token: failed to verify token:
token verification failed for all audiences: error for aud \"argo-cd\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-
known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>, error for aud \"argo-cd-cli\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-
known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>"
```

Using a self-signed certificate

First obtain the self-signed certificate used by the Deckhouse Platform identity and access subsystem:

```
d8 k -n d8-user-authn get secret ingress-tls -o jsonpath='{.data.tls\.crt}' | base64 -d
```

Then add the obtained certificate to the OIDC connector configuration in the `rootCAs` section of ArgoCD:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
            config:
              issuer: "https://dex.<CLUSTER_DOMAIN>/"
              rootCAs:
                - |
                  -----BEGIN CERTIFICATE-----
                  <Self-signed certificate obtained in the previous step>
                  -----END CERTIFICATE-----
              clientID: "dex-client-argocd@argocd"
              clientSecret: "$dex-client-argocd:clientSecret"
              insecureEnableGroups: true
              scopes:
                - profile
                - email
                - openid
                - groups
    provider: dex
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: argocd-ingress-tls
    insecure: true
```

Creating a user token

Argo CD does not allow issuing permanent tokens for users authenticated with SSO. At the same time, such users can use the `argocd` CLI utility by specifying the `--sso` flag during authentication:

```
argocd login <ARGOCD_DOMAIN>:443 --sso
```

When this command runs on the administrator workstation, a web browser opens with the Deckhouse Platform authentication form.

> **i** To issue a permanent token, create a local Argo CD user and set the `apiKey` attribute for them. In this case, the user has access rights only to the Argo CD API. For details, see Creating additional local users.

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` custom resources

- Module `user-authn` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring role-based access control

Argo CD uses its own role-based access control (RBAC) model, which is not based on the Kubernetes or Deckhouse Platform role model. The Argo CD role model lets you restrict access to resources and operations through its own policies and roles.

Before configuring role-based access control, complete authentication and authorization setup. After that, assign roles to users and groups and set permissions at the level of the entire Argo CD instance or individual projects with an AppProject object.

You can define role-based access control (RBAC) rules in two places:

- globally — in the ArgoCD object;
- at the project level — in the roles of an AppProject object.

Built-in roles

Argo CD has two predefined roles with a set of policies:

- `role:readonly` — read-only access (used by default for all authenticated users);
- `role:admin` — access with administrator privileges.

Full description of predefined role policies...

```
# Policies of the readonly role.
p, role:readonly, applications, get, */*, allow
p, role:readonly, applicationsets, get, */*, allow
p, role:readonly, certificates, get, *, allow
p, role:readonly, clusters, get, *, allow
p, role:readonly, repositories, get, *, allow
p, role:readonly, write-repositories, get, *, allow
p, role:readonly, projects, get, *, allow
p, role:readonly, accounts, get, *, allow
p, role:readonly, gpgkeys, get, *, allow
p, role:readonly, logs, get, */*, allow

# Policies of the admin role.
p, role:admin, applications, create, */*, allow
p, role:admin, applications, update, */*, allow
p, role:admin, applications, update/*, */*, allow
p, role:admin, applications, delete, */*, allow
p, role:admin, applications, delete/*, */*, allow
p, role:admin, applications, sync, */*, allow
p, role:admin, applications, override, */*, allow
p, role:admin, applications, action/*, */*, allow
p, role:admin, applicationsets, get, */*, allow
p, role:admin, applicationsets, create, */*, allow
p, role:admin, applicationsets, update, */*, allow
p, role:admin, applicationsets, delete, */*, allow
p, role:admin, certificates, create, *, allow
p, role:admin, certificates, update, *, allow
p, role:admin, certificates, delete, *, allow
p, role:admin, clusters, create, *, allow
p, role:admin, clusters, update, *, allow
p, role:admin, clusters, delete, *, allow
p, role:admin, repositories, create, *, allow
p, role:admin, repositories, update, *, allow
p, role:admin, repositories, delete, *, allow
p, role:admin, write-repositories, create, *, allow
p, role:admin, write-repositories, update, *, allow
p, role:admin, write-repositories, delete, *, allow
p, role:admin, projects, create, *, allow
p, role:admin, projects, update, *, allow
p, role:admin, projects, delete, *, allow
p, role:admin, accounts, update, *, allow
p, role:admin, gpgkeys, create, *, allow
p, role:admin, gpgkeys, delete, *, allow
p, role:admin, exec, create, */*, allow

# The admin role includes all privileges of the readonly role.
g, role:admin, role:readonly

# The local admin user is assigned the admin role.
g, admin, role:admin
```

Default policy for authenticated users

After successful authentication, the user receives the role specified in the `spec.rbac.defaultPolicy` parameter of ArgoCD (`role:readonly` by default).

> ⚠ All authenticated users receive at least the permissions defined in the `spec.rbac.defaultPolicy` parameter. These rights cannot be revoked with a `deny` effect rule.

Create a separate role, for example `role:authenticated`, grant it a minimal set of permissions, and use it in `spec.rbac.defaultPolicy`.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    defaultPolicy: role:authenticated
    policy: |
      p, role:authenticated, applications, get, */*, allow
...
```

Anonymous access

If you enable anonymous access to an Argo CD instance, users can receive rights according to the policy specified in the `spec.rbac.defaultPolicy` parameter without authentication.

Anonymous access is enabled with the `spec.usersAnonymousEnabled` parameter of ArgoCD.

> ⚠ When enabling anonymous access, create a separate default role, for example `role:unauthenticated`, and assign it in `spec.rbac.defaultPolicy`.

Role-based access control structure

The Argo CD role-based access control syntax is based on the Casbin model and uses two types of records:

- binding a user or group to a role;
- assigning permissions to a role, user, or group.

Role binding

Syntax:

```
g, <user/group>, <role>
```

Where:

- `<user/group>` — a local user, SSO user, or group;
- `<role>` — an internal Argo CD role.

Example:

```
g, my-org:team-beta, role:admin
g, user@example.org, role:readonly
```

Access policy

Syntax:

```
p, <role/user/group>, <resource>, <action>, <object>, <effect>
```

Where:

- `<role/user/group>` — the subject to which the rule is assigned;
- `<resource>` — the resource type;
- `<action>` — the allowed operation;
- `<object>` — the object to which the rule applies;
- `<effect>` — the check result: `allow` or `deny`.

Example:

```
p, role:developer, applications, sync, dev-project/*, allow
p, role:developer, logs, get, dev-project/*, allow
```

> **i** To assign rules to a group, first bind the group to a role with a `g, <group>, <role>` record. After that, assign permissions to the role itself.

Resources and supported actions

The following are the main Argo CD resources and actions that you can use in RBAC policies.

Resource	get	create	update	delete	sync	action	override	invoke
`applications`	✓	✓	✓	✓	✓	✓	✓	✓
`applicationsets`	✓	✓	✓	✓	✓	✓	✓	✓
`clusters`	✓	✓	✓	✓	✓	✓	✓	✓
`projects`	✓	✓	✓	✓	✓	✓	✓	✓
`repositories`	✓	✓	✓	✓	✓	✓	✓	✓
`accounts`	✓	✓	✓	✓	✓	✓	✓	✓
`certificates`	✓	✓	✓	✓	✓	✓	✓	✓
`gpgkeys`	✓	✓	✓	✓	✓	✓	✓	✓
`logs`	✓	✓	✓	✓	✓	✓	✓	✓
`exec`	✓	✓	✓	✓	✓	✓	✓	✓
`extensions`	✓	✓	✓	✓	✓	✓	✓	✓

Application-bound policies

Some resources are bound to specific applications:

- `applications`;
- `applicationsets`;
- `logs`;
- `exec`.

For such resources, the `<object>` value usually has the `<APP_PROJECT>/<APP_NAME>` format.

Example:

```
p, example-user, applications, get, *, allow
p, example-user, logs, get, example-project/my-app, allow
```

If Argo CD has application placement in arbitrary namespaces enabled, the `<object>` format changes to `<APP_PROJECT>/<APP_NAMESPACE>/<APP_NAME>`.

Example:

```
p, example-user, applications, get, */app-namespace/*, allow
p, example-user, logs, get, example-project/app-namespace/my-app, allow
```

The `applications` resource

The `applications` resource supports both basic actions and more detailed rules.

Detailed permissions for `update` and `delete`

The `update` and `delete` permissions granted for an application allow changing or deleting the

application itself, but not its nested resources.

To allow an operation on application resources, use the format:

```
<action>/<group>/<kind>/<namespace>/<name>
```

For example, to allow a user to delete only Pods in the `prod-app` application:

```
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

To allow updating all application resources but not the application itself:

```
p, example-user, applications, update/*, default/prod-app, allow
```

To deny deleting the application but allow deleting Pods:

```
p, example-user, applications, delete, default/prod-app, deny
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

To allow updating the application but deny updating any subresources:

```
p, example-user, applications, update, default/prod-app, allow
p, example-user, applications, update/*, default/prod-app, deny
```

⚠ Argo CD does not use the `/` character as a separator when comparing glob patterns. Therefore, always specify the full resource path, that is, use patterns with four `/` characters.

The `action` action

The `action` action corresponds to built-in or custom actions on application resources.

Format of the `<action>` value:

```
action/<group>/<kind>/<action-name>
```

For example:

- `action/extensions/DaemonSet/restart` — an action for `DaemonSet`;
- `action//Pod/maintenance-off` — an action for `Pod` that has no API group.

Policy example:

```
p, example-user, applications, action//Pod/maintenance-off, default/*, allow
p, example-user, applications, action/extensions/DaemonSet/*, default/*, allow
```

To allow all actions on application resources:

```
p, example-user, applications, action/*, default/*, allow
```

The `override` action

The `override` permission allows passing arbitrary manifests or another revision when synchronizing an `Application`.

⚠ The `override` right lets a user effectively change the composition or state of deployed application resources. Grant it only to users who truly need it.

If the `application.sync.requireOverridePrivilegeForRevisionSync: 'true'` parameter is enabled in the `argocd-cm` ConfigMap, passing an arbitrary revision during synchronization also requires the `override` right.

Other resources

The `applicationsets` resource

The `applicationsets` resource also belongs to application-bound policies. The `create` permission for this resource effectively allows creating Application objects through an ApplicationSet object.

Example:

```
p, dev-group, applicationsets, *, dev-project/*, allow
```

With this rule, users from `dev-group` can create an ApplicationSet that manages applications only within the `dev-project` project.

The `logs` resource

The `get` permission for the `logs` resource allows viewing Pod logs of an application through the Argo CD web interface. In meaning, this is equivalent to `d8 k logs`.

The `exec` resource

The `create` permission for the `exec` resource allows opening an `exec` session in an application Pod through the Argo CD web interface. In meaning, this is similar to `d8 k exec`.

The `extensions` resource

The `extensions` resource is used to invoke proxy extensions. Checks for such rules work together with the `applications` resource: the user must have read access to the application from which the request is made.

Example:

```
p, example-user, applications, get, default/*, allow
p, example-user, extensions, invoke, httpbin, allow
```

The `deny` effect

If a policy with the `deny` effect matches a request, access is denied even if there are more specific rules with the `allow` effect.

The order of lines in `spec.rbac.defaultPolicy` does not affect the result.

Policy evaluation order and matching modes

Access checks are performed in two stages:

1. Rules from `spec.rbac.defaultPolicy` are checked.
2. If no decision is found, rules related to the user and their groups are checked.

If access is explicitly allowed or denied by the default policy, further checks are not performed.

Argo CD supports two matching modes for values set in `spec.rbac.policyMatcherMode`:

- `glob` — matching by glob patterns;
- `regex` — matching by regular expressions.

Matching in `glob` mode

In `glob` mode, tokens are treated as whole strings without special separators.

Policy example:

```
p, example-user, applications, action/extensions/*, default/*, allow
```

If the `example-user` user invokes the `extensions/DaemonSet/test` action, the following checks are performed:

1. `example-user` matches `example-user`.
2. `applications` matches `applications`.
3. `action/extensions/DaemonSet/test` matches `action/extensions/*`.
4. `default/my-app` matches `default/*`.

Using SSO users and groups

The `scopes` parameter defines which OIDC scopes Argo CD must analyze during RBAC checks in addition to `sub`. If the parameter is not set, the default value is `'[groups]'`.

Example of an ArgoCD object with scopes configured:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    policy: |
      p, my-org:team-alpha, applications, sync, my-project/*, allow
      g, my-org:team-beta, role:admin
      g, user@example.org, role:admin
      g, admin, role:admin
      g, role:admin, role:readonly
    defaultPolicy: role:readonly
    scopes: '[groups, email]'
```

In this example:

1. `g, admin, role:admin` explicitly binds the built-in `admin` user to the `role:admin` role.
2. `g, role:admin, role:readonly` sets role inheritance: everyone who has `role:admin` automatically also gets `role:readonly` rights.

You can combine this approach with roles at the AppProject level. For example, you can create the `team-beta-project` project and assign rights to users and groups:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-beta-project
  namespace: argocd
spec:
  roles:
    - name: admin
      description: Administrator permissions for team-beta.
      policies:
        - p, proj:team-beta-project:admin, applications, *, team-beta-project/*, allow
      groups:
        - user@example.org
        - my-org:team-beta
```

Local users

You can grant access to local users in two ways:

- assign rules directly;
- bind the user to a role.

Example of assigning a rule directly:

```
p, my-local-user, applications, sync, my-project/*, allow
```

Example of binding to a role:

```
g, my-local-user, role:admin
```

⚠ If SSO and local users are used at the same time, ambiguity is possible. For example, if the local user `sally` is bound to `role:admin`, and one of the SSO scope values is also `sally`, that SSO user may also receive `role:admin` rights.

To avoid this situation, when using SSO and local users together, assign rules to local users directly rather than through roles.

Example:

```
p, my-local-user, *, *, *, allow
```

Testing RBAC policies

To verify RBAC rules, use the `argocd` CLI utility. To check whether a specific user, group, or role can perform a certain action, use the following command:

```
argocd admin settings rbac can
```

For example, run the command below to check whether the `admin@deckhouse.io` user has rights to create `applications` in the `default` project:

```
argocd admin settings rbac can admin@deckhouse.io create application 'default/*' --namespace argocd
```

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` custom resources

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Marketplace

Marketplace is a package management system in Deckhouse Platform (DP). It lets administrators connect package repositories, discover available packages, and make them available to project users to install applications.

Marketplace is available in DP starting from version 1.76.

For the Marketplace architecture, including core concepts, application development and configuration, templates, Nelm annotations, hooks, and the application lifecycle, refer to “Architecture”.

To learn about installing and removing applications from Marketplace, as well as troubleshooting, refer to “Usage”.

Administrator tasks

A cluster administrator:

1. Connects a package repository by creating a PackageRepository resource.
2. Monitors the repository scanning operations by Marketplace, discovering available packages.
3. Grants users the permissions to create Application objects in their namespaces. Any authenticated user can view package versions.
4. If needed, chooses the packages available in projects.

Users interact with packages through the Application object.

Marketplace resources

The following resources are used for working with the Marketplace:

Resource	Short name	Scope	Description
PackageRepository	—	Cluster	Package repository and its scanning settings
PackageRepositoryOperation	`pro`	Cluster	A repository scanning operation
ApplicationPackageVersion	`apv`	Cluster	Discovered version of a package
ApplicationPackage	`ap`	Cluster	Package summary: the repositories it's available in, the versions the release channels point to, and the applications that use it
Application	—	Namespace	Installed application instance managed by a user

The Marketplace documentation includes the following sections:

- "Package repositories" describes how to connect a repository and check its status.
- "Scanning" describes how to monitor scanning operations and trigger a scanning manually.
- "Configuring package access" describes how to configure access to packages in projects.

Additional resources

- Using Marketplace
- Global custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Package repositories

To connect Deckhouse Platform (DP) to an application package repository, use the PackageRepository resource. Once the resource is created, DP scans the repository and creates an ApplicationPackageVersion object for each discovered package version.

Example of a PackageRepository manifest:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepository
metadata:
  name: my-registry
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <BASE64_ENCODED_DOCKER_CONFIG>
```

Authentication and scan interval

Authentication

To authenticate to the package repository, use one of the following methods:

- `dockerCfg`: Docker configuration in the `~/.docker/config.json` format, Base64-encoded. It

must contain an `auths` entry for the repository host. DP uses the username and password from this entry.

- `login` and `password`: Explicit credentials. If both methods are specified, `login` and `password` take precedence.

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    login: my-user
    password: my-password
```

If the repository uses a self-signed TLS certificate, specify the CA certificate in the `ca` parameter:

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <BASE64_ENCODED_DOCKER_CONFIG>
    ca: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

Scan interval

By default, DP scans the repository **every 6 hours**. To change the interval, set the `scanInterval` parameter. The minimum interval is 3 minutes:

```
spec:
  registry:
    repo: registry.example.com/packages
  scanInterval: 1h30m
```

For details about starting a scanning or viewing results, refer to “Scanning”.

Checking repository status

The state of the repository is shown in the status of the PackageRepository object.

To display brief information about the status, use the following command:

```
d8 k get packagerepository <REPOSITORY_NAME>
```

Output columns:

Column	Description
`Phase`	Repository phase. Displayed as `Active` after the first successful scan
`Scan`	Time of the last successful scan
`Repository`	Repository address (`spec.registry.repo`)
`Packages`	Number of packages in the repository
`MSG`	Message of the `LastScanSucceeded` condition, for example, the error of the last scan

For detailed information about the status, use the following command:

```
d8 k get packagerepository <REPOSITORY_NAME> -o yaml
```

For status field descriptions, refer to the PackageRepository documentation.

To check the result of the last scan, view the `LastScanSucceeded` condition:

```
d8 k get packagerepository <REPOSITORY_NAME> \
  -o jsonpath='{.status.conditions[?(@.type=="LastScanSucceeded")]}'
```

If the last scan completed successfully, the condition status is set to `True`. Otherwise, it is `False`, and the error details are available in the status message.

Viewing discovered package versions

After a successful scan, ApplicationPackageVersion objects (or `apv` for short) are automatically created in the cluster.

To print out a list of all created ApplicationPackageVersion objects, run the following command:

```
d8 k get apv
```

Example output:

```
NAME                           PACKAGE    REPOSITORY    METADATALOADED   USEDBY
AGE
my-registry-redis-v7.2.0       redis      my-registry   True                      5m
my-registry-postgres-v15.0.0   postgres   my-registry   True                      5m
```

To filter the versions by package name, use the following command (the example shows the versions of the `redis` package):

```
d8 k get apv -l packages.deckhouse.io/package=redis
```

> i `MetadataLoaded=True` means that the package's OpenAPI schema, description, and requirements were successfully loaded from the repository. A package version with `MetadataLoaded=False` cannot be installed until the metadata is loaded.

Additional resources

- Global custom resources

> i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Scanning

Deckhouse Platform (DP) uses PackageRepositoryOperation objects to configure package repository scanning. Each scan operation discovers new package versions and creates ApplicationPackageVersion objects for them. DP creates operations automatically, but you can also create them manually.

Viewing scan operations

To view a list of scan operations, use the following command (you can use `pro` instead of PackageRepositoryOperation):

```
d8 k get pro
```

DP keeps the 10 most recent operations of each repository and deletes operations if that threshold is exceeded, starting from the oldest ones.

Example output:

```
NAME                   COUNT   COMPLETED   MSG   COMPLETIONTIME
test-scan-1780052895   23      True              3h38m
test-scan-1780053890   23      True              3h22m
```

Output columns:

Column	Description
`Count`	Total number of packages found during the scan
`Completed`	Whether the operation has finished (`True` or `False`). A failed operation is considered completed too. The result is defined by the reason of the `Completed` condition: `ScanSucceeded` or `ScanFailed`
`MSG`	Message of the `Completed` condition, for example, the scan error
`CompletionTime`	Time when the operation completed

To filter operations by repository, use the following command:

```
d8 k get pro -l packages.deckhouse.io/repository=<REPOSITORY_NAME>
```

Inspecting a scan operation

For full scan results including per-package details, use the following command:

```
d8 k get pro <OPERATION_NAME> -o yaml
```

For status field descriptions, refer to the PackageRepositoryOperation documentation.

An example command for viewing packages with errors:

```
d8 k get pro <OPERATION_NAME> \
  -o jsonpath='{range .status.packages.failed[*]}{.name}: {range .errors[*]}
{.version} - {.message}{"\n"}{end}{end}'
```

Triggering a manual scan

DP starts a scan of a PackageRepository when the resource is created, when its specification is modified, when DP restarts, and then every `spec.scanInterval` (6 hours by default, at least 3 minutes). A scan is skipped if the previous operation of the repository hasn't completed yet.

To start the scanning immediately, create a PackageRepositoryOperation manually, for example:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepositoryOperation
metadata:
  generateName: my-registry-scan-manual-
spec:
  packageRepositoryName: my-registry
  type: Update
  update:
    fullScan: true
```

> **i** The `generateName` field gives every operation a unique name. It works only with `d8 k create`, but not with `d8 k apply`.

Alternatively, to create a scan operation, use the following command:

```
d8 system package scan <REPOSITORY_NAME>
```

This command creates a PackageRepositoryOperation with `spec.type: Update` and `spec.update.fullScan: true`.

Full scanning

To run a full scanning of a package repository, set the `spec.update.fullScan: true` parameter in a PackageRepositoryOperation. In this mode, all available package versions are scanned, not only the ones higher than discovered previously.

With `spec.update.fullScan: false` (by default), only the SemVer tags with a version higher than the package version already processed in the cluster are processed.

Automatic operations perform a full repository scan until the first successful scan. After that, automatic scans process only versions newer than the latest package version already processed.

Use the full scanning in the following cases:

- A package version was published after a newer version, for example, a patch for a previous minor version.
- ApplicationPackageVersion resources are missing from the cluster for versions available in the package repository.

Additional resources

- Global custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring package access

By default, a user who may create Application objects in a namespace can install applications from any package. If the `multitenancy-manager` module is enabled, the cluster administrator can restrict the packages available for installation in each project and each namespace of a project. A request to create an Application with unavailable package will be rejected.

To configure access to packages, Deckhouse Platform registers an ApplicationPackage as a cluster-wide resource named `applicationpackages`. The `multitenancy-manager` module checks the `spec.packageName` field value of Application objects in project namespaces. Package access in configured using the ClusterResourceGrantPolicy resource. A policy does not grant RBAC permissions: to install an application, a user still needs a role that allows creating Application objects in the corresponding namespace.

For details on the access management mechanism, refer to the `multitenancy-manager` module documentation.

Allowing packages in a project

To allow installing only the `postgres` and `redis` packages in the `my-project` project, create the following ClusterResourceGrantPolicy:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: my-project-packages
spec:
  projectSelector:
    matchLabels:
      projects.deckhouse.io/project: my-project
  resources:
    - resourceName: applicationpackages
      allowed:
        - postgres
        - redis
```

The names in `allowed` field are the names of ApplicationPackage objects corresponding to the respective `spec.packageName` field values of an Application. To list the available packages, run `d8 k get ap`.

The `multitenancy-manager` module automatically assigns the `projects.deckhouse.io/project: <PROJECT_NAME>` label to every Project, which makes the policy applied to every namespace in the respective namespace, including the additional namespaces created after the policy is created.

To apply a policy to several projects, use one of the labels:

- `projects.deckhouse.io/project-template: <PROJECT_TEMPLATE_NAME>` label, which selects all projects based on the template
- your own label on the Project objects, for example `team: payments`, which selects a custom group of projects.

Once a policy with `allowed` field applies to a project, the packages that are not listed in it become unavailable for installation. If several policies apply to a project, their `allowed` lists are combined.

Allowing packages in one namespace of a project

Using the `namespaceSelector` parameter, you can restrict a policy to selected namespaces of a project. The policy is applied only to namespaces whose labels match the selector. It does not restrict the other namespaces of the project. If no other policy restricts them, every package is available there.

For example, to allow installation of the `postgres` package only in the `my-project-db` namespace of the `my-project` project and deny installation of any packages in other namespaces of the project, create two policies:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: my-project-no-packages
spec:
  projectSelector:
    matchLabels:
      projects.deckhouse.io/project: my-project
  resources:
    - resourceName: applicationpackages
      availabilityDefault: None
---
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: my-project-db-packages
spec:
  projectSelector:
    matchLabels:
      projects.deckhouse.io/project: my-project
  namespaceSelector:
    matchLabels:
      kubernetes.io/metadata.name: my-project-db
  resources:
    - resourceName: applicationpackages
      allowed:
        - postgres
```

The first policy (`my-project-no-packages`) makes every package unavailable for installation in all namespaces of the project. The second one (`my-project-db-packages`) allows the `postgres` packages in the `my-project-db` namespace. If a package is allowed at least by a single policy, it's available for installation, even if another policy sets `availabilityDefault: None`.

The `NewNamespacesCovered` condition in the status of the second policy is `False`, because a new additional namespace of the project does not match its `namespaceSelector`. However the first policy covers such namespaces, so no packages are available for installation in them.

Denying packages

To deny installation of specific packages while keeping the others available, list them in `denied`.

The following is an example policy that makes the `experimental-db` package unavailable in every project:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: deny-experimental-db
spec:
  projectSelector: {}
  resources:
    - resourceName: applicationpackages
      denied:
        - experimental-db
```

The empty `projectSelector` parameter selects every project. A package listed under `denied` is unavailable for installation even if another policy allows it.

Packages in a project template or in a project

Package access rules can be set in `spec.resources` of a project template, so that they apply to every project created from the template, or in `spec.grants.resources` of a single Project.

For example, to allow installation of only `postgres` and `redis` packages in the projects of a template,

add the following entry to the template:

```
spec:
  resources:
    - resourceName: applicationpackages
      allowed:
        - postgres
        - redis
```

For details, refer to “Granting cluster-wide resources through a project template” and “Granting cluster-wide resources to one project”.

Checking the available packages

The packages available for installation in a namespace are listed in the `applicationpackages` AvailableClusterResource of the corresponding namespace. That same list can be read by the project users.

To view the list, run the following command:

```
d8 k get available applicationpackages -n <NAMESPACE> -o yaml
```

Requests of the platform components and of members of the `system:masters` group are not checked against policies. To check on restrictions, use a project user account.

For example, to perform a server-side dry run of an Application manifest without creating the object, run the following:

```
d8 k apply -f application.yaml --dry-run=server --as <USER_NAME>
```

Applications installed before a restriction

Introducing policy restrictions does not remove installed applications. When an existing Application object is changed, the availability of its package is checked only if the package is modified. Therefore an application installed before its package became unavailable to the project keeps working.

Such applications are counted by the `d8 cluster_objects_grant_violated` metric, and the `ClusterResourceGrantPolicyViolation` alert fires. The alert stops when the package becomes available to the project again or the corresponding Application object is deleted.

Limitations

When configuring access to packages, consider the following limitations:

- Policies apply only in project namespaces. They do not apply to the namespaces of the virtual projects `default` and `deckhouse`, so any package can be installed there.
- A policy selects available packages by name. It does not restrict the repository an available package is installed from (`spec.packageRepositoryName`) or its version.
- A policy does not hide unavailable packages. Any authenticated user can read the ApplicationPackageVersion objects of all packages, including the ones unavailable to the project. The `status.usedBy` field of a version lists the applications installed from it and their namespaces in all projects.
- While the controller of the `multitenancy-manager` module is unavailable, Application objects cannot be created or changed in project namespaces. To keep the controller available, enable `highAvailability` in the `multitenancy-manager` settings.

Additional resources

- Global custom resources
- Module `multitenancy-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Platform update

Deckhouse Platform (DP) includes a built-in update management mechanism for both the platform itself and Kubernetes cluster components. To roll out new versions consistently and safely, DP uses a five-channel update system ranging from the newest and unstable (Alpha) to the most thoroughly tested (Rock Solid). Each new version gradually moves through these channels, helping identify issues early and ensuring stability in production environments. For details on each channel, refer to the Architecture section.

> ⚠ DP does not support version downgrades. After a release is installed, rolling the platform back to an earlier minor or patch version is not possible.
>
> When switching to a more stable channel, the platform remains on the current release until a newer version becomes available on that channel.
>
> Rolling back by one minor version is possible only for the Kubernetes (control plane) version, not for the DP version.

You can configure the update process. Both automatic and manual modes are supported, as well as update windows and other parameters. See Update configuration for details.

An alerting system is used to track the update status and quickly report any issues. For more information, refer to Notification settings.

The following features are supported:

- Release notifications: DP can send release notifications via webhooks.
- Retrieving the changelog: Each DP release includes a changelog, available both in the cluster and as part of the release notification.
- Checking dependencies before update: Checks for component dependencies before proceeding with an update to prevent conflicts.

For more details on updating DP in air-gapped environment, see the “Updating DP in air-gapped environment” guide.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring updates

Deckhouse Platform (DP) supports a flexible update mechanism, allowing you to select release channels and configure the update mode. Release channels help you balance stability with the speed of receiving new features.

The update mode configuration lets you choose between automatic or manual updates and define update windows during which new versions can be installed. Together, these features help you avoid updates at inconvenient times and control migration to new releases.

> Up-to-date information about DP versions available on different release channels is available at releases.deckhouse.io.

Checking the current release channel

To check which release channel is used in your cluster, run the following command:

```
d8 k get mc deckhouse -o yaml | grep releaseChannel
```

Example output:

```
    releaseChannel: Stable
```

Switching release channels

To switch the release channel, specify the new channel in the `settings.releaseChannel` parameter of the `deckhouse` module.

Example configuration using the `Stable` channel:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
```

Update modes

DP supports two update modes that determine how new versions are applied:

- **Automatic mode** — the cluster updates as soon as a new version appears on the selected release channel. Two configuration options are available:
  - `AutoPatch` — only patch version updates within the current minor version are applied;
  - `Auto` — both patch and minor versions are applied automatically.
- **Manual mode (`Manual`)** — both patch and minor versions are applied only after manual approval.

For automatic mode, update windows settings are available — allowed updates are applied taking `update.windows` into account, if windows are configured. If no windows are set, the update is applied as soon as the version appears on the release channel.

> **i**
>
> The `update` settings of the `deckhouse` module (mode and windows) apply by default to both built-in and external modules.
>
> If a module has no dedicated update policy (ModuleUpdatePolicy) and its ModuleConfig does not specify the `updatePolicy` parameter, the module inherits `releaseChannel` and `update` from the `deckhouse` ModuleConfig.

Patch versions only within the current minor (AutoPatch)

To keep DP updates within the current minor version (apply patch versions only), use the `AutoPatch` mode.

For example, if version `v1.70.1` is installed, DP can automatically update to `v1.70.2`, but will not move to `v1.71.*` without manual approval.

This is the default value. To set the `AutoPatch` mode explicitly, run the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
{"mode":"AutoPatch"}}}}'
```

To approve a minor version update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DP version:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

Automatic updates for all versions (Auto)

In the `Auto` mode, DP automatically applies both patch and minor versions, taking update windows into account if they are configured.

To enable the `Auto` mode, the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Auto"}}}}'
```

Manual mode (Manual)

In the `Manual` mode, DP receives information about new versions in the cluster, but applying both patch and minor versions requires manual approval.

To enable the `Manual` mode, the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
{"mode":"Manual"}}}}'
```

Checking the current update mode

To determine the current update mode used in the cluster, inspect the configuration of the `deckhouse` module with the following command:

```
d8 k get mc deckhouse -o yaml
```

Example output:

```
spec:
  settings:
    releaseChannel: Stable
    update:
      mode: AutoPatch
      windows:
      - days:
        - Mon
        from: "19:00"
        to: "20:00"
```

How automatic updates are applied

Automatic update mode is enabled when the `releaseChannel` parameter is specified in the `deckhouse` module configuration. When this condition is met:

1. DP checks the release channel every minute for new releases.
2. When a new release appears, DP downloads it into the cluster and creates a DeckhouseRelease custom resource.
3. Once the DeckhouseRelease resource appears in the cluster, DP applies the corresponding update according to the configured update settings (by default — automatically, at any time).

To view the list and status of all releases in the cluster, run the following command:

```
d8 k get deckhousereleases
```

⚠ Starting from DP 1.70, patch version updates (for example, an update to version `1.70.2` when version `1.70.1` is installed) are applied taking update windows into account. Prior to DP 1.70, patch version updates are applied without regard to the update mode and windows.

Release pinning

Release pinning refers to fully or partially disabling automatic updates.

There are three ways to restrict automatic updates in Deckhouse:

- Enable manual update approval mode.

  In this mode, DP will receive updates into the cluster, but applying patch and minor versions will require manual approval.

  To enable manual update approval mode, set the `settings.update.mode` parameter to `Manual` in the `deckhouse` module configuration using the following command:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
  {"mode":"Manual"}}}}'
  ```

  To approve an update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DP version:

  ```
  d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
  ```

- Enable automatic updates for patch versions only.

  In this mode, DP will receive updates into the cluster, but applying minor versions will require manual approval. Patch versions within the current minor version will be applied automatically, taking update windows into account if they are configured.

  For example, if you have DP version `v1.70.1` installed, after enabling this mode, Deckhouse can automatically update to `v1.70.2`, but it will not update to `v1.71.*` without manual approval.

  To enable automatic updates for patch versions only, set the `settings.update.mode` parameter to `AutoPatch` in the `deckhouse` module configuration using the following command:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
  {"mode":"AutoPatch"}}}}'
  ```

  To approve a minor version update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DP version:

  ```
  d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
  ```

- Manually set the target DP version tag for the `deckhouse` Deployment and remove the `releaseChannel` parameter from the `deckhouse` module configuration.

In this case, DP will remain at the specified version, and no information about newer available versions (DeckhouseRelease objects) will appear in the cluster.

> **i** **Important**. This mode blocks the installation of patch releases, which may include critical security or bug fixes. If you need to receive patches within the current minor version, use the `AutoPatch` mode instead of hard pinning.

Example of pinning version `v1.66.3` for DP EE and removing the `releaseChannel` parameter from the `deckhouse` module configuration:

```
d8 k -ti -n d8-system exec svc/deckhouse-leader -c deckhouse -- kubectl set image
deployment/deckhouse deckhouse=registry.deckhouse.io/deckhouse/ee:v1.66.3
```

```
d8 k patch mc deckhouse --type=json -p='[{"op": "remove", "path":
"/spec/settings/releaseChannel"}]'
```

Manual update approval

Manual approval of DP updates is required in the following cases:

- The DP update confirmation mode is enabled.

  This means the `settings.update.mode` parameter of the `deckhouse` module is set to either `Manual` (confirmation required for both patch and minor updates) or `AutoPatch` (confirmation required only for minor updates).

To approve an update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target version:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

- Automatic update approval is disabled for a NodeGroup, for updates that might cause temporary downtime of system components.

  This means the `spec.disruptions.approvalMode` parameter of the corresponding NodeGroup resource is set to `Manual`.

  To apply the update, set the `update.node.deckhouse.io/disruption-approved=` annotation on each node in the group:

  Example:

  ```
  d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
  ```

Update windows

DP allows you to define *update windows*, which are specific time intervals during which automatic updates are allowed. Using update windows ensures that updates won't be installed at inconvenient times or during periods of high cluster load.

Applying updates when update windows are configured

- If update windows are configured, DP installs new versions only during the specified windows.
- If no update windows are configured, the update is applied as soon as a new version appears on the configured release channel.

Configuring update windows

You can manage DP update windows in the following ways:

- **To control general updates**, use the `update.windows` parameter in the `deckhouse` module configuration;
- **To control updates that may lead to short-term downtime of system components**, use the `disruptions.automatic.windows` and `disruptions.rollingUpdate.windows` parameters in the NodeGroup resource.

Configuration examples

- Two daily update windows: from 08:00 to 10:00 and from 20:00 to 22:00 (UTC):

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse
  spec:
    version: 1
    settings:
      releaseChannel: EarlyAccess
      update:
        windows:
          - from: "8:00"
            to: "10:00"
          - from: "20:00"
            to: "22:00"
  ```

- Update windows on Tuesdays and Saturdays from 18:00 to 19:30 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
          days:
            - Tue
            - Sat
```

Updating virtualization

If virtualization is enabled in the cluster, its components fall into two categories that are updated differently:

- the components that manage virtualization resources (the control layer);
- the components that run virtual machines (the “firmware”).

Updating the control layer doesn’t affect the virtual machines that are already running, but it drops open VNC and serial console connections while the component restarts.

Updating the “firmware” may require migrating virtual machines to the new version. DKP migrates a machine once, and if the migration fails, the machine owner has to move or restart it manually. To pick a suitable time for such an update, use update windows and the manual mode.

Additional resources

- Global custom resources
- Module `deckhouse` documentation
- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuring notifications on new releases

Deckhouse Platform (DP) generates alerts in the monitoring system and can automatically send notifications about upcoming minor updates to external systems. This helps you plan updates and prepare for them in advance.

Conditions for sending notifications to external systems:

- DP is operating in automatic update mode.
- A minor version update is planned (notifications are not sent for patch versions).
- A webhook for notifications is configured.

Alerts in the monitoring system

If an update requires making changes to the cluster (for example, updating the Kubernetes or OS version), DP generates designated alerts. These alerts include:

- `D8NodeHasDeprecatedOSVersion`: Nodes with an unsupported OS version have been detected in the cluster.
- `HelmReleasesHasResourcesWithDeprecatedVersions`: Some Helm releases use deprecated resources.
- `D8KubernetesVersionIsDeprecated`: The installed Kubernetes version is no longer supported.

If any of these alerts appear, make sure to resolve them before updating. This helps avoid disruptions and ensures the cluster remains stable after the update.

Configuring notifications

In the `Auto` update mode, you can configure a webhook call to receive a notification about an upcoming minor DP version update.

Additionally, notifications are generated not only for Deckhouse updates but also for updates of any modules, including individual ones.
In some cases, the system may initiate multiple notifications simultaneously (10–20 notifications) at approximately 15-second intervals.

> **i** Notifications are available only in the `Auto` update mode; they are not generated in the `Manual` mode.

> **i** Specifying a webhook is optional: if the `update.notification.webhook` parameter is not set but the `update.notification.minimalNotificationTime` parameter is specified, the update will still be postponed for the defined duration. In this case, the appearance of the DeckhouseRelease resource in the cluster, named after the new version, can be considered the notification.

After a new minor DP version appears in the selected update channel but before it is applied in the cluster, a POST request will be sent to the configured webhook address.

The minimalNotificationTime parameter allows postponing the update installation for a defined period, providing time to react to the notification while respecting update windows.
If the webhook is unavailable, each failed attempt to send the notification will postpone the update by the same amount, which may lead to the update being indefinitely deferred.

> **⚠** If your webhook returns any status code out of 2xx range, DP retries sending the notification up to five times with exponential backoff. If all attempts fail, the release is blocked until the webhook becomes available again.

For easier error handling and debugging, when returning error codes the webhook should return a JSON response with the following structure:

- `code` — optional internal error code for programmatic handling;
- `message` — a human-readable description of what went wrong.

If the webhook returns a successful HTTP status (2xx), DP treats the notification as successful regardless of the response body.

Minimal Go webhook example...

```
package main
import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
)
// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
  Subject      string          `json:"subject"`
  ModuleName   string          `json:"moduleName,omitempty"`
  Version      string          `json:"version"`
  Requirements  map[string]string `json:"requirements,omitempty"`
  ChangelogLink string          `json:"changelogLink,omitempty"`
  ApplyTime    string          `json:"applyTime,omitempty"`
  Message      string          `json:"message"`
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
```

```
  Code    string `json:"code,omitempty"`
  Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
  if r.Method != http.MethodPost {
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  defer r.Body.Close()

  var data WebhookData
  if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
    log.Printf("failed to decode payload: %v", err)
    w.WriteHeader(http.StatusInternalServerError)
    return
  }

  // Print payload fields
  log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
    data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
  log.Printf("message=%s", data.Message)

  // Example conditional logic: fail intentionally for testing
  if data.Version == "v0.0.0-fail" {
    // Return structured error response with error status code
    errorResp := ResponseError{
      Code:    "TEST_FAILURE",
      Message: "intentional failure for testing",
    }

    w.WriteHeader(http.StatusBadRequest)
    json.NewEncoder(w).Encode(errorResp)
    return
  }

  // Return success response with 2xx status code
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Notification processed successfully"))
}

func main() {
  mux := http.NewServeMux()
  mux.HandleFunc("/webhook", handler)

  addr := ":8080"
  fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
  if err := http.ListenAndServe(addr, mux); err != nil {
    log.Fatal(err)
  }
}
```

Supported notifications parameters

- `update.notification.webhook`: URL for sending notifications. A POST request with update information is sent as soon as a new minor version appears on the selected release channel, but before the update is installed in the cluster.
- `update.notification.auth`: Authentication parameters for calling the webhook. If this parameter is not set, authentication is not used.
  - `update.notification.auth.basic`: Basic authentication. The username and password are passed in the `Authorization` header as `Basic <base64(username:password)>`.
    - `update.notification.auth.basic.username`: Username.
    - `update.notification.auth.basic.password`: Password.
  - `update.notification.auth.bearerToken`: Token-based authentication. The token is passed in the `Authorization` header as `Bearer <token>`.

- `update.notification.minimalNotificationTime`: Minimum interval between the appearance of a new minor version on the selected release channel and the start of the update. Specified in hours and minutes: `30m`, `1h`, `2h30m`, `24h`. If an update window is configured, the update will be applied only after the `minimalNotificationTime` has elapsed and within the defined window.
- `update.notification.tlsSkipVerify`: Disables TLS certificate verification when calling the webhook (for example, if a self-signed certificate is used). Set to `false` by default.

Example `update.notification` configuration using basic authentication:

```
update:
  notification:
    webhook: https://release-webhook.mydomain.com
    minimalNotificationTime: 4h
    auth:
      basic:
        username: myusername
        password: mypassword
    tlsSkipVerify: true
```

Example `update.notification` configuration without authentication:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h
```

Notification format

When the conditions for sending notifications are met, DP sends a POST request to the specified webhook with the header `Content-Type: application/json`.

Example request body for Deckhouse release:

```
{
  "subject": "Deckhouse",
  "version": "1.68",
  "requirements":  {"k8s": "1.29.0"},
  "changelogLink":
"https://github.com/deckhouse/deckhouse/blob/main/CHANGELOG/CHANGELOG-v1.68.md",
  "applyTime": "2025-02-01T14:30:00Z00:00",
  "message": "New Deckhouse Release 1.68 is available. Release will be applied at:
Wednesday, 05-Feb-25 14:30:00 UTC"
}
```

Example request body for module release:

```
{
  "subject": "Module",
  "moduleName": "test",
  "version": "0.7.14",
  "requirements": {"deckhouse": ">= 1.69"},
  "changelogLink":
"https://github.com/deckhouse/modules/blob/main/CHANGELOG/CHANGELOG-test-
v0.7.14.md",
  "applyTime": "2025-12-17T12:32:24Z",
  "message": "New test Release 0.7.14 is available. Release will be applied at:
Wednesday, 17-Dec-25 12:32:24 UTC"
}
```


Field descriptions:

- `subject`: Release subject (“Deckhouse” for Deckhouse releases or “Module” for module releases).
- `moduleName`: Module name (present only for module releases).
- `version`: Minor version number (a string).
- `requirements`: Object with requirements for the new version (for example, the minimum Kubernetes version).
- `changelogLink`: Link to the changelog describing changes in the new minor version.
- `applyTime`: Scheduled update date and time in RFC3339 format with the configured update windows considered.
- `message`: A short text notification about the available minor version and its scheduled installation time.

Additional resources

- Global custom resources
- Module `deckhouse` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Updating Kubernetes and versioning

Updating and version management

The control plane update process in DP is fully automated.

- DP supports the latest five Kubernetes versions.
- You can roll back the control plane one minor version and upgrade forward several minor versions — one at a time.
- Patch versions (e.g., `1.35.3` → `1.35.5`) are updated automatically with Deckhouse and cannot be managed manually.
- Minor versions are set manually using the `kubernetesVersion` parameter in the `control-plane-manager` ModuleConfig.

Changing the Kubernetes version

You can downgrade only one minor version from the highest version ever used in the cluster. Upgrades proceed one minor version at a time.

1. Open the `control-plane-manager` ModuleConfig editor:

   ```
   d8 k edit mc control-plane-manager
   ```

2. Set the target Kubernetes version using the `settings.kubernetesVersion` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "1.35"
```

> **i** The kubernetesVersion field in ClusterConfiguration, previously used to select the Kubernetes version, has been deprecated. Until the `settings.kubernetesVersion` parameter of the `control-plane-manager` module is set, this field determines the cluster's Kubernetes version. Once the parameter is set, its value is used instead. Do not change the Kubernetes version via ClusterConfiguration.

3. Save the changes.
4. Wait for the update to complete. You can track the update progress with the `d8 k get no` command. The update can be considered complete when the updated version appears in the `VERSION` column of each cluster node in the command output.

Monitoring Kubernetes update progress

The `control-plane-manager` module includes the `update-observer` component, which provides up-to-date information about the Kubernetes version update process in the cluster.

`update-observer` component:

- reads cluster configuration from the `d8-cluster-configuration` Secret
- tracks kubelet versions on all nodes via `nodeInfo.kubeletVersion`
- collects versions from all control plane instances via the `control-plane-manager.deckhouse.io/kubernetes-version` annotation
- creates and maintains the **`d8-cluster-kubernetes`** ConfigMap in the `kube-system` namespace with detailed update status.

The `d8-cluster-kubernetes` ConfigMap displays:

- **Component status**: Versions of control plane components (kube-apiserver, kube-scheduler, kube-controller-manager) on each master node.
- **Node progress**: How many nodes have been updated and the total count.
- **Target and current version**: The desired version from configuration and the actual state during the update.
- **Version mismatch**: If any components are running a version different from the target (including newer than desired).
- **Version lists**:
  - `supportedVersions`: Lists minor Kubernetes versions supported in the current DP release.
  - `availableVersions`: Lists versions that can be selected for upgrade or downgrade in the *current* cluster. The set is limited by the highest minor version ever installed on the cluster and by the rule that downgrade proceeds one minor at a time.
- **`maxUsedKubernetesVersion`**: The highest minor version the cluster has ever converged onto. It only ever rises, and it is what limits how far the version can be rolled back.
- **`automaticVersion`**: Minor version that will be used when the update mode is `Automatic`. It describes the installed DP build rather than the state of the cluster.

During `ControlPlaneUpdating`, `status.progress` reflects overall upgrade progress across intermediate minor versions. For a multi-hop upgrade (for example, 1.33 → 1.35), the percentage increases as each hop completes, not only when every control plane component reaches the final target.

Minor versions in the ConfigMap (`spec`, `status`, and metadata labels such as `k8s-version` and `max-k8s-version`) use the same string format as the `kubernetesVersion` parameter, meaning without a `v` prefix (for example, `"1.33"`).

You can see in real time which components are being updated, at what stage the process is, and whether the update has "stuck" on any node or component.

To view the update status, run the command:

```
d8 k get configmap d8-cluster-kubernetes -n kube-system -o yaml
```

ConfigMap content examples

The `data.spec` and `data.status` fields store YAML with the `spec` field (target version and update mode) and the `status` field (current state). Below are examples of the content for various situations.

Cluster up to date (3 master nodes, 3 worker nodes)

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.32"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    phase: UpToDate
    controlPlane:
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-2
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-0
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: idle
    lastReconciliationTime: "2026-02-02T01:13:05Z"
    lastUpToDateTime: "2026-01-30T16:26:36Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.32"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "20837731"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```


Start of update (e.g., Kubernetes version downgrade)

The target version is already set; the control plane or nodes are still transitioning to it.

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    phase: ControlPlaneUpdating
    progress: 0%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:34:42Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21379847"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Control plane update in progress

Some master nodes are already on the new version, others are still updating; progress is shown as a

percentage:

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-2
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:41:55Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21388343"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Intermediate step of a multi-hop upgrade (e.g. 1.33 → 1.35)

In the configuration, the target Kubernetes version may be several minor versions ahead of the current cluster version. In this case, the upgrade is performed step by step through intermediate minor versions.

The `status.currentVersion` field reflects the current upgrade step (the active minor version the cluster is being brought to at the moment). Within a single step, individual control plane components may temporarily run different minor versions.

The `progress` field reflects the entire upgrade path, including intermediate versions. Therefore, its value can be significantly greater than 0% even before all components reach the target version.

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.35"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.34"
    supportedVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    - "1.35"
    availableVersions:
    - "1.33"
    - "1.34"
    - "1.35"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.35"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.34"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.34"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-04-17T09:43:08Z"
    lastUpToDateTime: "2026-04-15T13:43:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "1753458"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Cluster up to date (2 master nodes, 1 arbitr node and 3 worker nodes)

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.33"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    phase: UpToDate
    controlPlane:
    - name: master-0
      phase: UpToDate
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-02-02T11:09:59Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21357074"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Failure of one or more control plane components

The master node has `phase: Failed`; the `description` field contains the reason (e.g., pod or container not in `Running` state):

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
    automaticVersion: "1.33"
  status: |
    currentVersion: "1.32"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    phase: ControlPlaneUpdating
    progress: 73%
    controlPlane:
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-0
      phase: Failed
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
```

Additional resources

- Global custom resources
- Module `control-plane-manager` documentation

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Switching between DP editions

This guide describes the steps required to switch the Deckhouse Platform edition in a running cluster.

Follow the sections in order.

The switching process differs depending on how you work with the image registry. Choose the method that applies to your cluster and follow the instructions.

A valid license key is required when switching to DP BE/SE/SE+/EE. It is not required when switching to DP CE.

> ⚠ This guide assumes the use of a public container image registry (`registry.deckhouse.io`). If you use a different registry address, adjust the commands or refer to the guide for switching Deckhouse to a third-party container image registry.
>
> Execute all commands on the master node of the existing cluster as the `root` user.

Pre-switch preparation

Before switching between revisions, follow these steps:

1. Make sure that the DP queues are empty.
2. Determine the current DP revision and version.
3. Verify that you can switch from the current version to the desired one.

Queue check

Make sure the DP queues are empty and there are no running tasks that could interfere with the switch:

```
d8 system queue list
```

Example output (queues are empty)...

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

Determining the current edition and version

To ensure the correctness of further steps, determine the current DP edition used in the cluster. This helps avoid errors during the switch and confirms that the required modules and features are supported in the new edition.

You can find the edition and version currently used in the cluster on the main page of the DP web interface, or by using CLI commands:

- edition:

  ```
  d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller global values -o yaml | yq '.deckhouseEdition'
  ```

- version:

  ```
  d8 k -n d8-system get deploy deckhouse -ojson | jq -r '.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print $NF}'
  ```

Checking whether switching to the desired edition is possible

Different DP editions support different sets of modules, Kubernetes versions, and features. It is important to understand what functional changes will occur during the switch and which capabilities will become unavailable. This will help you prepare for the switching process.

A comparison of DP editions by module set can be found in the documentation on the Edition comparison page.

What to consider before switching:

- To DP CE

- To DP BE
- To DP SE
- To DP SE+
- To DP EE

1. Determine the list of internal modules used in the cluster that are not supported in DP new edition. To do this, follow these steps:

   1. Get the list of internal modules not supported in DP ce:

      ```
      (set -e
      trap 'echo "Execution error"' ERR

      echo
      echo "Running the deckhouse pod of the new edition with the command 'sleep
      -- infinity'"

      DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
      '.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
      | awk -F: '{print $NF}')
      d8 k run ce-image --image=registry.deckhouse.io/deckhouse/ce/install:$DECKHOUSE_VERSION
      \
      --command sleep -- infinity
      d8 k wait --for=condition=ready pod/ce-image --timeout=300s

      echo
      echo "Getting information about internal modules from the current and new
      edition"
      echo "Comparing internal modules of the current and new edition"
      NEW_MODULES=$(d8 k exec ce-image -- ls -l deckhouse/modules/ |  grep -oE "\d.*-\w*"
      | awk {'print $9'} | cut -c5-)
      USED_MODULES=$(d8 k get modules -o custom-
      columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.ph
      ase | grep Embedded | grep -E 'Enabled|Ready' | awk {'print $1'})
      MODULES_TO_DISABLE=$(echo "$USED_MODULES" | tr ' ' '\n' | grep -Fxv -f <(echo
      "$NEW_MODULES" | tr ' ' '\n') 2>&1) || { status=$?; [ $status -eq 1 ] &&
      MODULES_TO_DISABLE="" || exit $status; }

      echo
      echo "Modules not supported in the desired edition (edition code - ce,
      version - $DECKHOUSE_VERSION):"
      echo MODULES_TO_DISABLE=\"$MODULES_TO_DISABLE\"
      )

      echo
      echo "Deleting the deckhouse pod of the new edition"
      d8 k delete pod/ce-image secret/ce-image-pull-secret --wait=true --ignore-not-
      found=true
      ```

   2. Disable the modules from the list if acceptable (the module functionality is not used, or you are ready to give it up). Otherwise, **abort the switching process.**

      You can disable the modules from the list in the DP web interface under System → System Management → Deckhouse → Modules, or by running the following command:

      ```
      echo $MODULES_TO_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module
      disable",$1'} | bash
      ```

2. Determine the list of external modules launched via `moduleSource/deckhouse` that are not supported in DP new edition. To do this, follow these steps:

   1. Get the list of external modules launched via `moduleSource/deckhouse` with the result of checking their availability in DP new edition:

```
(set -e
trap 'echo "Execution error"' ERR
d8-crane() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr "$@"
   else
       crane "$@"
   fi
}


check_external_modules() {
   local repo="$1"

   local modules=$(d8 k get modulerelease -o json | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.kind=="ModuleSource" and
.name=="deckhouse")) | "\(.spec.moduleName) \(.spec.version)"')
   [[ -z "$modules" ]] && return 0
   local registry_modules=$(d8-crane ls "$repo" 2>/dev/null)

   echo "Module | In registry"
   echo "-------|------------"

   echo "$modules" | while read module version; do
       grep -qx "$module" <<< "$registry_modules" && list="" || list=""
     echo "$module | $list"
   done
}

d8-crane --help > /dev/null 2>&1 || { echo "ERROR: crane not found"; exit 1; }
check_external_modules "registry.deckhouse.io/deckhouse/ce/modules"
)
```

2. Disable the modules that were not found in the new edition if acceptable. Otherwise, **abort the switching process.**

3. Make sure all tasks in the DP queue are complete before continuing the switching process:

   ```
   d8 system queue list
   ```

   Example output (queues are empty)...

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

1. Determine the list of internal modules used in the cluster that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of internal modules not supported in DP be:

```
(set -e
trap 'echo "Execution error"' ERR

echo
echo "Running the deckhouse pod of the new edition with the command 'sleep
-- infinity'"

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F: '{print $NF}')
d8 k create secret docker-registry be-image-pull-secret --docker-
server=registry.deckhouse.io --docker-username=license-token --docker-
password=${LICENSE_TOKEN}
d8 k run be-image --image=registry.deckhouse.io/deckhouse/be/install:$DECKHOUSE_VERSION
\
--overrides="{\"spec\": {\"imagePullSecrets\":[{\"name\": \"be-image-pull-
secret\"}]}}" \
--command sleep -- infinity
d8 k wait --for=condition=ready pod/be-image --timeout=300s

echo
echo "Getting information about internal modules from the current and new
edition"
echo "Comparing internal modules of the current and new edition"
NEW_MODULES=$(d8 k exec be-image -- ls -l deckhouse/modules/ |  grep -oE "\d.*-\w*"
| awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.ph
ase | grep Embedded | grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_TO_DISABLE=$(echo "$USED_MODULES" | tr ' ' '\n' | grep -Fxv -f <(echo
"$NEW_MODULES" | tr ' ' '\n') 2>&1) || { status=$?; [ $status -eq 1 ] &&
MODULES_TO_DISABLE="" || exit $status; }

echo
echo "Modules not supported in the desired edition (edition code - be,
version - $DECKHOUSE_VERSION):"
echo MODULES_TO_DISABLE=\"$MODULES_TO_DISABLE\"
)

echo
echo "Deleting the deckhouse pod of the new edition"
d8 k delete pod/be-image secret/be-image-pull-secret --wait=true --ignore-not-
found=true
```

3. Disable the modules from the list if acceptable (the module functionality is not used, or you are ready to give it up). Otherwise, **abort the switching process.**

   You can disable the modules from the list in the DP web interface under System → System Management → Deckhouse → Modules, or by running the following command:

   ```
   echo $MODULES_TO_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module
   disable",$1'} | bash
   ```

2. Determine the list of external modules launched via `moduleSource/deckhouse` that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of external modules launched via `moduleSource/deckhouse` with the result of checking their availability in DP new edition:

```
(set -e
trap 'echo "Execution error"' ERR
d8-crane() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr "$@"
   else
       crane "$@"
   fi
}

d8-crane-login() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr login "$1" -u "license-token" -p "$2"
   else
       crane auth login "$1" -u "license-token" -p "$2"
   fi
}

check_external_modules() {
   local repo="$1"

   local modules=$(d8 k get modulerelease -o json | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.kind=="ModuleSource" and
.name=="deckhouse")) | "\(.spec.moduleName) \(.spec.version)"')
   [[ -z "$modules" ]] && return 0
   local registry_modules=$(d8-crane ls "$repo" 2>/dev/null)

   echo "Module | In registry"
   echo "-------|------------"

   echo "$modules" | while read module version; do
       grep -qx "$module" <<< "$registry_modules" && list="" || list=""
     echo "$module | $list"
   done
}

d8-crane --help > /dev/null 2>&1 || { echo "ERROR: crane not found"; exit 1; }
d8-crane-login "registry.deckhouse.io" "$LICENSE_TOKEN"
check_external_modules "registry.deckhouse.io/deckhouse/be/modules"
)
```

3. Disable the modules that were not found in the new edition if acceptable. Otherwise, **abort the switching process.**

3. Make sure all tasks in the DP queue are complete before continuing the switching process:

   ```
   d8 system queue list
   ```

   Example output (queues are empty)...

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

1. Determine the list of internal modules used in the cluster that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of internal modules not supported in DP se:

```
(set -e
trap 'echo "Execution error"' ERR

echo
echo "Running the deckhouse pod of the new edition with the command 'sleep
-- infinity'"

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F: '{print $NF}')
d8 k create secret docker-registry se-image-pull-secret --docker-
server=registry.deckhouse.io --docker-username=license-token --docker-
password=${LICENSE_TOKEN}
d8 k run se-image --image=registry.deckhouse.io/deckhouse/se/install:$DECKHOUSE_VERSION
\
--overrides="{\"spec\": {\"imagePullSecrets\":[{\"name\": \"se-image-pull-
secret\"}]}}" \
--command sleep -- infinity
d8 k wait --for=condition=ready pod/se-image --timeout=300s

echo
echo "Getting information about internal modules from the current and new
edition"
echo "Comparing internal modules of the current and new edition"
NEW_MODULES=$(d8 k exec se-image -- ls -l deckhouse/modules/ |  grep -oE "\d.*-\w*"
| awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.ph
ase | grep Embedded | grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_TO_DISABLE=$(echo "$USED_MODULES" | tr ' ' '\n' | grep -Fxv -f <(echo
"$NEW_MODULES" | tr ' ' '\n') 2>&1) || { status=$?; [ $status -eq 1 ] &&
MODULES_TO_DISABLE="" || exit $status; }

echo
echo "Modules not supported in the desired edition (edition code - se,
version - $DECKHOUSE_VERSION):"
echo MODULES_TO_DISABLE=\"$MODULES_TO_DISABLE\"
)

echo
echo "Deleting the deckhouse pod of the new edition"
d8 k delete pod/se-image secret/se-image-pull-secret --wait=true --ignore-not-
found=true
```

3. Disable the modules from the list if acceptable (the module functionality is not used, or you are ready to give it up). Otherwise, **abort the switching process.**

   You can disable the modules from the list in the DP web interface under System → System Management → Deckhouse → Modules, or by running the following command:

   ```
   echo $MODULES_TO_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module
   disable",$1'} | bash
   ```

2. Determine the list of external modules launched via `moduleSource/deckhouse` that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of external modules launched via `moduleSource/deckhouse` with the result of checking their availability in DP new edition:

```
(set -e
trap 'echo "Execution error"' ERR
d8-crane() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr "$@"
   else
       crane "$@"
   fi
}

d8-crane-login() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr login "$1" -u "license-token" -p "$2"
   else
       crane auth login "$1" -u "license-token" -p "$2"
   fi
}

check_external_modules() {
   local repo="$1"

   local modules=$(d8 k get modulerelease -o json | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.kind=="ModuleSource" and
.name=="deckhouse")) | "\(.spec.moduleName) \(.spec.version)"')
   [[ -z "$modules" ]] && return 0
   local registry_modules=$(d8-crane ls "$repo" 2>/dev/null)

   echo "Module | In registry"
   echo "-------|------------"

   echo "$modules" | while read module version; do
       grep -qx "$module" <<< "$registry_modules" && list="" || list=""
     echo "$module | $list"
   done
}

d8-crane --help > /dev/null 2>&1 || { echo "ERROR: crane not found"; exit 1; }
d8-crane-login "registry.deckhouse.io" "$LICENSE_TOKEN"
check_external_modules "registry.deckhouse.io/deckhouse/se/modules"
)
```

3. Disable the modules that were not found in the new edition if acceptable. Otherwise, **abort the switching process.**

3. Make sure all tasks in the DP queue are complete before continuing the switching process:

   ```
   d8 system queue list
   ```

   Example output (queues are empty)...

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

1. Determine the list of internal modules used in the cluster that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of internal modules not supported in DP se-plus:

```
(set -e
trap 'echo "Execution error"' ERR

echo
echo "Running the deckhouse pod of the new edition with the command 'sleep
-- infinity'"

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F: '{print $NF}')
d8 k create secret docker-registry se-plus-image-pull-secret --docker-
server=registry.deckhouse.io --docker-username=license-token --docker-
password=${LICENSE_TOKEN}
d8 k run se-plus-image --image=registry.deckhouse.io/deckhouse/se-
plus/install:$DECKHOUSE_VERSION \
--overrides="{\"spec\": {\"imagePullSecrets\":[{\"name\": \"se-plus-image-
pull-secret\"}]}}" \
--command sleep -- infinity
d8 k wait --for=condition=ready pod/se-plus-image --timeout=300s

echo
echo "Getting information about internal modules from the current and new
edition"
echo "Comparing internal modules of the current and new edition"
NEW_MODULES=$(d8 k exec se-plus-image -- ls -l deckhouse/modules/ |  grep -oE
"\d.*-\w*" | awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.ph
ase | grep Embedded | grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_TO_DISABLE=$(echo "$USED_MODULES" | tr ' ' '\n' | grep -Fxv -f <(echo
"$NEW_MODULES" | tr ' ' '\n') 2>&1) || { status=$?; [ $status -eq 1 ] &&
MODULES_TO_DISABLE="" || exit $status; }

echo
echo "Modules not supported in the desired edition (edition code - se-plus,
version - $DECKHOUSE_VERSION):"
echo MODULES_TO_DISABLE=\"$MODULES_TO_DISABLE\"
)

echo
echo "Deleting the deckhouse pod of the new edition"
d8 k delete pod/se-plus-image secret/se-plus-image-pull-secret --wait=true --ignore-not-
found=true
```

3. Disable the modules from the list if acceptable (the module functionality is not used, or you are ready to give it up). Otherwise, **abort the switching process.**

   You can disable the modules from the list in the DP web interface under System → System Management → Deckhouse → Modules, or by running the following command:

   ```
   echo $MODULES_TO_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module
   disable",$1'} | bash
   ```

2. Determine the list of external modules launched via `moduleSource/deckhouse` that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of external modules launched via `moduleSource/deckhouse` with the result of checking their availability in DP new edition:

```
(set -e
trap 'echo "Execution error"' ERR
d8-crane() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr "$@"
   else
       crane "$@"
   fi
}

d8-crane-login() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr login "$1" -u "license-token" -p "$2"
   else
       crane auth login "$1" -u "license-token" -p "$2"
   fi
}

check_external_modules() {
   local repo="$1"

   local modules=$(d8 k get modulerelease -o json | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.kind=="ModuleSource" and
.name=="deckhouse")) | "\(.spec.moduleName) \(.spec.version)"')
   [[ -z "$modules" ]] && return 0
   local registry_modules=$(d8-crane ls "$repo" 2>/dev/null)

   echo "Module | In registry"
   echo "-------|------------"

   echo "$modules" | while read module version; do
       grep -qx "$module" <<< "$registry_modules" && list="" || list=""
     echo "$module | $list"
   done
}

d8-crane --help > /dev/null 2>&1 || { echo "ERROR: crane not found"; exit 1; }
d8-crane-login "registry.deckhouse.io" "$LICENSE_TOKEN"
check_external_modules "registry.deckhouse.io/deckhouse/se-plus/modules"
)
```

3. Disable the modules that were not found in the new edition if acceptable. Otherwise, **abort the switching process.**

3. Make sure all tasks in the DP queue are complete before continuing the switching process:

   ```
   d8 system queue list
   ```

   Example output (queues are empty)...

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

1. Determine the list of internal modules used in the cluster that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of internal modules not supported in DP ee:

```
(set -e
trap 'echo "Execution error"' ERR

echo
echo "Running the deckhouse pod of the new edition with the command 'sleep
-- infinity'"

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F: '{print $NF}')
d8 k create secret docker-registry ee-image-pull-secret --docker-
server=registry.deckhouse.io --docker-username=license-token --docker-
password=${LICENSE_TOKEN}
d8 k run ee-image --image=registry.deckhouse.io/deckhouse/ee/install:$DECKHOUSE_VERSION
\
--overrides="{\"spec\": {\"imagePullSecrets\":[{\"name\": \"ee-image-pull-
secret\"}]}}" \
--command sleep -- infinity
d8 k wait --for=condition=ready pod/ee-image --timeout=300s

echo
echo "Getting information about internal modules from the current and new
edition"
echo "Comparing internal modules of the current and new edition"
NEW_MODULES=$(d8 k exec ee-image -- ls -l deckhouse/modules/ |  grep -oE "\d.*-\w*"
| awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.ph
ase | grep Embedded | grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_TO_DISABLE=$(echo "$USED_MODULES" | tr ' ' '\n' | grep -Fxv -f <(echo
"$NEW_MODULES" | tr ' ' '\n') 2>&1) || { status=$?; [ $status -eq 1 ] &&
MODULES_TO_DISABLE="" || exit $status; }

echo
echo "Modules not supported in the desired edition (edition code - ee,
version - $DECKHOUSE_VERSION):"
echo MODULES_TO_DISABLE=\"$MODULES_TO_DISABLE\"
)

echo
echo "Deleting the deckhouse pod of the new edition"
d8 k delete pod/ee-image secret/ee-image-pull-secret --wait=true --ignore-not-
found=true
```

3. Disable the modules from the list if acceptable (the module functionality is not used, or you are ready to give it up). Otherwise, **abort the switching process.**

   You can disable the modules from the list in the DP web interface under System → System Management → Deckhouse → Modules, or by running the following command:

   ```
   echo $MODULES_TO_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module
   disable",$1'} | bash
   ```

2. Determine the list of external modules launched via `moduleSource/deckhouse` that are not supported in DP new edition. To do this, follow these steps:

   1. Set the environment variable with the license key for the edition you plan to switch to:

      ```
      LICENSE_TOKEN=<LICENSE_KEY>
      ```

   2. Get the list of external modules launched via `moduleSource/deckhouse` with the result of checking their availability in DP new edition:

```
(set -e
trap 'echo "Execution error"' ERR
d8-crane() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr "$@"
   else
       crane "$@"
   fi
}

d8-crane-login() {
   if command -v d8 &>/dev/null && d8 cr &>/dev/null 2>&1; then
       d8 cr login "$1" -u "license-token" -p "$2"
   else
       crane auth login "$1" -u "license-token" -p "$2"
   fi
}

check_external_modules() {
   local repo="$1"

   local modules=$(d8 k get modulerelease -o json | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.kind=="ModuleSource" and
.name=="deckhouse")) | "\(.spec.moduleName) \(.spec.version)"')
   [[ -z "$modules" ]] && return 0
   local registry_modules=$(d8-crane ls "$repo" 2>/dev/null)

   echo "Module | In registry"
   echo "-------|------------"

   echo "$modules" | while read module version; do
       grep -qx "$module" <<< "$registry_modules" && list="" || list=""
     echo "$module | $list"
   done
}

d8-crane --help > /dev/null 2>&1 || { echo "ERROR: crane not found"; exit 1; }
d8-crane-login "registry.deckhouse.io" "$LICENSE_TOKEN"
check_external_modules "registry.deckhouse.io/deckhouse/ee/modules"
)
```

3. Disable the modules that were not found in the new edition if acceptable. Otherwise, **abort the switching process.**

3. Make sure all tasks in the DP queue are complete before continuing the switching process:

   ```
   d8 system queue list
   ```

   Example output (queues are empty)...

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

Switching the edition

Selecting a switching method

When selecting a switching method, consider how the DP container image repository is configured within the cluster.

There are two ways to work with the DP container image registry:

- Using the `registry` module — **(recommended)** — the configuration for working with the DP registry is specified in the `registry` section of the `deckhouse` module parameters (ModuleConfig `deckhouse`). This method provides a smoother transition process and automatic verification that the required images are present. If your cluster uses this method for working with the DP container registry, use the section “Switching Using the registry Module” to switch editions.
- Without using the `registry` module — the configuration for working with the DP registry is set during cluster installation in `InitConfiguration`, and the `registry.mode` of the `deckhouse` module (ModuleConfig `deckhouse`) is set to `Unmanaged`, and the `registry.unmanaged` parameter of the `deckhouse` module is not specified.

  This method is the only one available for managed Kubernetes clusters where the control plane is managed by a cloud provider rather than DP (e.g. Amazon EKS, Azure AKS, Google GKE, etc.).

  If your cluster uses this method for managing the DP container image repository, refer to the section “Switching without using the registry module” to switch versions.

Switching using the registry module

⚠

- Before proceeding, complete the preparatory steps described in the Pre-switch preparation section.
- This switching method must be performed only in `Unmanaged` mode. Make sure that the cluster uses the `registry` module. ModuleConfig `deckhouse` must contain the registry parameters of the previous DP edition in `Unmanaged` mode. If this is not the case, use the “Switching without the registry module” section: DP 1.78 and later no longer contain the v1 implementation of the `registry` module, so a cluster cannot be moved to it.
- This method is not applicable for managed Kubernetes (EKS, AKS, GKE).

1. In ModuleConfig `deckhouse`, set `imagesRepo` to the target edition and `checkMode: Relax`:

   Run the command to edit ModuleConfig `deckhouse`:

   ```
   d8 k edit moduleconfig deckhouse
   ```

   Choose the example for your target edition:

   - DP CE
   - DP BE
   - DP SE
   - DP SE+
   - DP EE

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
         unmanaged:
           checkMode: Relax
           imagesRepo: registry.deckhouse.io/deckhouse/ce
           scheme: HTTPS
   ```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        license: <LICENSE_KEY>
        checkMode: Relax
        imagesRepo: registry.deckhouse.io/deckhouse/be
        scheme: HTTPS
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        license: <LICENSE_KEY>
        checkMode: Relax
        imagesRepo: registry.deckhouse.io/deckhouse/se
        scheme: HTTPS
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        license: <LICENSE_KEY>
        checkMode: Relax
        imagesRepo: registry.deckhouse.io/deckhouse/se-plus
        scheme: HTTPS
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        license: <LICENSE_KEY>
        checkMode: Relax
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
```

2. Make sure that `deckhouse` pods has started after the registry change:
   1. To do this, run the command:

      ```
      d8 k -n d8-system get pods -l "app=deckhouse"
      ```

   2. If Deckhouse is in ImagePullBackoff with an error downloading sidecar containers, specify the actual images:

      Get the actual digests of the sidecar images:

      ```
      DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
      '.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
      | awk -F':' '{print $1}')
      DKP_TAG=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
      '.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
      | awk -F':' '{print $2}')

      d8 k run dkp-image --image=$DECKHOUSE_VERSION/install:$DKP_TAG --command sleep --
      infinity
      d8 k wait --for=condition=ready pod/dkp-image --timeout=300s

      DECKHOUSE_KUBE_RBAC_PROXY=$(d8 k exec dkp-image -- cat
      /deckhouse/candi/images_digests.json | jq -r ".common.kubeRbacProxy")
      DECKHOUSE_INIT=$(d8 k exec dkp-image -- cat /deckhouse/candi/images_digests.json | jq -
      r ".deckhouse.init")

      d8 k delete pod dkp-image
      ```

      Apply images:

      ```
      d8 k -n d8-system set image deployment/deckhouse \
      kube-rbac-proxy=$DECKHOUSE_VERSION@$DECKHOUSE_KUBE_RBAC_PROXY \
      init-downloaded-modules=$DECKHOUSE_VERSION@$DECKHOUSE_INIT
      ```

   3. Make sure there is no image download error:

      ```
      d8 k -n d8-system get pods -l "app=deckhouse"
      ```

3. Wait for the switch to complete.

   Check the status:

```
d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
map_values(@base64d) | .conditions = (.conditions | from_yaml) | {"conditions":
[.conditions[] | select(.type == "Ready" or .type ==
"RegistryContainsRequiredImages")]}'
```

Example of a successful output:

```
conditions:
  - lastTransitionTime: "2026-05-05T13:53:23Z"
    message: |-
      Mode: Default
      registry.deckhouse.io: all 1 items are checked
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
  - lastTransitionTime: "2026-05-05T13:54:49Z"
    message: ""
    reason: ""
    status: "True"
    type: Ready
```

4. Set `checkMode` back to `Default`:

```
d8 k patch moduleconfig deckhouse --type=json -p='[{"op": "replace", "path":
"/spec/settings/registry/unmanaged/checkMode", "value": "Default"}]'
```

5. Check the switch status again.

   Check the status:

```
d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
map_values(@base64d) | .conditions = (.conditions | from_yaml) | {"conditions":
[.conditions[] | select(.type == "Ready" or .type ==
"RegistryContainsRequiredImages")]}'
```

   Example of a successful output:

```
conditions:
  - lastTransitionTime: "2026-05-05T13:53:23Z"
    message: |-
      Mode: Default
      registry.deckhouse.io: all 182 items are checked
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
  - lastTransitionTime: "2026-05-05T13:54:49Z"
    message: ""
    reason: ""
    status: "True"
    type: Ready
```

6. Check for pods with image pull errors:

```
d8 k get pods -A | awk 'NR==1 || /^d8-/' | grep -E 'ImagePullBackOff|ErrImagePull'
```

   For each problematic module, run the following commands **on all master nodes**, specifying the module name:

```
rm -rf /var/lib/deckhouse/downloaded/<MODULE_NAME>/
d8 k rollout restart deploy -n d8-system deckhouse
```

7. Check for pods using the old registry:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] |
select(.image | contains("deckhouse.io/deckhouse/<PREVIOUS_EDITION_CODE>"))) |
.metadata.namespace + "\t" + .metadata.name' | sort | uniq
```

Switching without the registry module

⚠

- Before proceeding, complete the preparatory steps described in the Pre-switch preparation section.
- Before applying, make sure that the `registry` module is not used in the cluster. ModuleConfig `deckhouse` must not contain registry parameters. The `registry` module must be disabled. If this is not the case, perform the migration to the deprecated registry management format (without the registry module).

Choose the target edition:

- DP CE/BE/SE/SE+/EE

1. Run the command to set the authentication credentials for the image registry:

   - DP CE
   - DP BE
   - DP SE
   - DP SE+
   - DP EE

   Not required for DP CE.

   i If you need to add configuration for an additional registry in containerd, refer to the How to add configuration for an additional registry in containerd section.

   For ContainerdV1:

```
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv1-be-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/be-registry.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.deckhouse.io"]
              endpoint = ["https://registry.deckhouse.io"]

[plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.io".a
uth]
            auth = "$AUTH_STRING"
    EOF
EOF
```

For ContainerdV2:

```
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv2-be-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 31
  content: |
    mkdir -p /etc/containerd/registry.d/registry.deckhouse.io
    bb-sync-file "/etc/containerd/registry.d/registry.deckhouse.io/hosts.toml"
- << "EOF"
    [host]

      [host."https://registry.deckhouse.io"]
         capabilities = ["pull", "resolve"]
         [host."https://registry.deckhouse.io".auth]
            username = "license-token"
            password = "${LICENSE_TOKEN}"
    EOF
EOF
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

> **i** If you need to add configuration for an additional registry in containerd, refer to the How to add configuration for an additional registry in containerd section.

For ContainerdV1:

```
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv1-se-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/se-registry.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.deckhouse.io"]
              endpoint = ["https://registry.deckhouse.io"]

[plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.io".a
uth]
            auth = "$AUTH_STRING"
    EOF
EOF
```

For ContainerdV2:

```
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv2-se-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 31
  content: |
    mkdir -p /etc/containerd/registry.d/registry.deckhouse.io
    bb-sync-file "/etc/containerd/registry.d/registry.deckhouse.io/hosts.toml"
- << "EOF"
    [host]

      [host."https://registry.deckhouse.io"]
         capabilities = ["pull", "resolve"]
         [host."https://registry.deckhouse.io".auth]
            username = "license-token"
            password = "${LICENSE_TOKEN}"
    EOF
EOF
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

> **i** If you need to add configuration for an additional registry in containerd, refer to the How to add configuration for an additional registry in containerd section.

For ContainerdV1:

```
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv1-se-plus-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/se-plus-registry.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.deckhouse.io"]
              endpoint = ["https://registry.deckhouse.io"]

[plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.io".a
uth]
            auth = "$AUTH_STRING"
    EOF
EOF
```

For ContainerdV2:

```
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv2-se-plus-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 31
  content: |
    mkdir -p /etc/containerd/registry.d/registry.deckhouse.io
    bb-sync-file "/etc/containerd/registry.d/registry.deckhouse.io/hosts.toml"
- << "EOF"
    [host]

      [host."https://registry.deckhouse.io"]
         capabilities = ["pull", "resolve"]
         [host."https://registry.deckhouse.io".auth]
            username = "license-token"
            password = "${LICENSE_TOKEN}"
    EOF
EOF
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

> **i** If you need to add configuration for an additional registry in containerd, refer to the How to add configuration for an additional registry in containerd section.

For ContainerdV1:

```
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv1-ee-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/ee-registry.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.deckhouse.io"]
              endpoint = ["https://registry.deckhouse.io"]

[plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.io".a
uth]
            auth = "$AUTH_STRING"
    EOF
EOF
```

For ContainerdV2:

```
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerdv2-ee-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 31
  content: |
    mkdir -p /etc/containerd/registry.d/registry.deckhouse.io
    bb-sync-file "/etc/containerd/registry.d/registry.deckhouse.io/hosts.toml"
- << "EOF"
    [host]

      [host."https://registry.deckhouse.io"]
         capabilities = ["pull", "resolve"]
         [host."https://registry.deckhouse.io".auth]
            username = "license-token"
            password = "${LICENSE_TOKEN}"
    EOF
EOF
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

2. Switch the container image registry:

   - DP CE
   - DP BE
   - DP SE
   - DP SE+
   - DP EE

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk
-F: '{print $NF}')

d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry \
--new-deckhouse-tag=$DECKHOUSE_VERSION \
registry.deckhouse.io/deckhouse/ce
```

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk
-F: '{print $NF}')

AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.io\":
{\"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\":
\"${AUTH_STRING}\"}}}" | base64 -w 0)

d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry \
--type merge \
--patch="{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"

d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry \
--user=license-token \
--password=$LICENSE_TOKEN \
--new-deckhouse-tag=$DECKHOUSE_VERSION \
registry.deckhouse.io/deckhouse/be
```

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk
-F: '{print $NF}')

AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.io\":
{\"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\":
\"${AUTH_STRING}\"}}}" | base64 -w 0)

d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry \
--type merge \
--patch="{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"

d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry \
--user=license-token \
--password=$LICENSE_TOKEN \
--new-deckhouse-tag=$DECKHOUSE_VERSION \
registry.deckhouse.io/deckhouse/se
```

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk
-F: '{print $NF}')

AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.io\":
{\"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\":
\"${AUTH_STRING}\"}}}" | base64 -w 0)

d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry \
--type merge \
--patch="{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"

d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry \
--user=license-token \
--password=$LICENSE_TOKEN \
--new-deckhouse-tag=$DECKHOUSE_VERSION \
registry.deckhouse.io/deckhouse/se-plus
```

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk
-F: '{print $NF}')

AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.io\":
{\"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\":
\"${AUTH_STRING}\"}}}" | base64 -w 0)

d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry \
--type merge \
--patch="{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"

d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper
change-registry \
--user=license-token \
--password=$LICENSE_TOKEN \
--new-deckhouse-tag=$DECKHOUSE_VERSION \
registry.deckhouse.io/deckhouse/ee
```

3. Make sure that `deckhouse` pods has started after the registry change:
   1. To do this, run the command:

      ```
      d8 k -n d8-system get pods -l "app=deckhouse"
      ```

   2. If Deckhouse is in ImagePullBackoff with an error downloading sidecar containers, specify the actual images:

      Get the actual digests of the sidecar images:

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F':' '{print $1}')
DKP_TAG=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image'
| awk -F':' '{print $2}')

d8 k run dkp-image --image=$DECKHOUSE_VERSION/install:$DKP_TAG --command sleep --
infinity
d8 k wait --for=condition=ready pod/dkp-image --timeout=300s

DECKHOUSE_KUBE_RBAC_PROXY=$(d8 k exec dkp-image -- cat
/deckhouse/candi/images_digests.json | jq -r ".common.kubeRbacProxy")
DECKHOUSE_INIT=$(d8 k exec dkp-image -- cat /deckhouse/candi/images_digests.json | jq -
r ".deckhouse.init")

d8 k delete pod dkp-image
```

Apply images:

```
d8 k -n d8-system set image deployment/deckhouse \
kube-rbac-proxy=$DECKHOUSE_VERSION@$DECKHOUSE_KUBE_RBAC_PROXY \
init-downloaded-modules=$DECKHOUSE_VERSION@$DECKHOUSE_INIT
```

3. Make sure there is no image download error:

```
d8 k -n d8-system get pods -l "app=deckhouse"
```

4. Wait for DP to be ready:

```
d8 system queue list
```

Example output (queues are empty)...

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

5. Check for pods with image pull errors:

```
d8 k get pods -A | awk 'NR==1 || /^d8-/' | grep -E 'ImagePullBackOff|ErrImagePull'
```

For each problematic module, run the following commands **on all master nodes**, specifying the module name:

```
rm -rf /var/lib/deckhouse/downloaded/<MODULE_NAME>/
d8 k rollout restart deploy -n d8-system deckhouse
```

6. Check for pods using the old registry:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] |
select(.image | contains("deckhouse.io/deckhouse/<PREVIOUS_EDITION_CODE>"))) |
.metadata.namespace + "\t" + .metadata.name' | sort | uniq
```

7. Perform cleanup:

   - DP CE
   - DP BE
   - DP SE
   - DP SE+
   - DP EE

Not required for DP CE.

For ContainerdV1

```
d8 k --as=system:sudouser delete ngc containerdv1-be-config.sh
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: del-temp-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    if [ -f /etc/containerd/conf.d/be-registry.toml ]; then
      rm -f /etc/containerd/conf.d/be-registry.toml
    fi
EOF
```

```
d8 k --as=system:sudouser delete ngc del-temp-config.sh
```

For ContainerdV2

```
d8 k --as=system:sudouser delete ngc containerdv2-be-config.sh
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

For ContainerdV1

```
d8 k --as=system:sudouser delete ngc containerdv1-se-config.sh
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: del-temp-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    if [ -f /etc/containerd/conf.d/se-registry.toml ]; then
      rm -f /etc/containerd/conf.d/se-registry.toml
    fi
EOF
```

```
d8 k --as=system:sudouser delete ngc del-temp-config.sh
```

For ContainerdV2

```
d8 k --as=system:sudouser delete ngc containerdv2-se-config.sh
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

For ContainerdV1

```
d8 k --as=system:sudouser delete ngc containerdv1-se-plus-config.sh
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: del-temp-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    if [ -f /etc/containerd/conf.d/se-plus-registry.toml ]; then
      rm -f /etc/containerd/conf.d/se-plus-registry.toml
    fi
EOF
```

```
d8 k --as=system:sudouser delete ngc del-temp-config.sh
```

For ContainerdV2

```
d8 k --as=system:sudouser delete ngc containerdv2-se-plus-config.sh
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

For ContainerdV1

```
d8 k --as=system:sudouser delete ngc containerdv1-ee-config.sh
d8 k --as=system:sudouser apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: del-temp-config.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 1
  content: |
    if [ -f /etc/containerd/conf.d/ee-registry.toml ]; then
      rm -f /etc/containerd/conf.d/ee-registry.toml
    fi
EOF
```

```
d8 k --as=system:sudouser delete ngc del-temp-config.sh
```

For ContainerdV2

```
d8 k --as=system:sudouser delete ngc containerdv2-ee-config.sh
```

Wait for the bashible service to synchronize (the `UPTODATE` column value for a NodeGroup must match `NODES`):

```
d8 k get ng -o custom-
columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate
```

The bashible log should contain `Configuration is in sync, nothing to do`:

```
journalctl -u bashible -n 5
```

Additional resources

- Global custom resources
- Module `deckhouse` documentation
- Module `node-manager` documentation
- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Uninstallation

Deleting a cluster deployed with a cloud provider

Perform the following steps to delete a cluster deployed with a cloud provider

1. Find out the release channel set in the cluster. To do this, run the command:

   ```
   kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
   ```

2. Run the Deckhouse installer:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
  registry.deckhouse.io/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

where:

- `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
- `<DECKHOUSE_REVISION>` — the Deckhouse edition (e. g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
- `<RELEASE CHANNEL>` — the Deckhouse release channel in kebab-case:
  - `alpha` — for the *Alpha* release channel;
  - `beta` — for the *Beta* release channel;
  - `early-access` — for the *Early Access* release channel;
  - `stable` — for the *Stable* release channel;
  - `rock-solid` — for the *Rock Solid* release channel.

Below is an example command to run the Deckhouse CE installer container:

```
docker run -it --pull=always \
  -v "$PWD/dhctl-tmp:/tmp/dhctl" \
  -v "$HOME/.ssh/:/tmp/.ssh/" registry.deckhouse.io/deckhouse/ce/install:stable bash
```

3. In the container you have started, run the following command:

```
dhctl destroy --ssh-user=<USER> \
  --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
  --yes-i-am-sane-and-i-understand-what-i-am-doing \
  --ssh-host=<MASTER_IP>
```

where:

- `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to perform actions via `sudo`, add the `--ask-become-pass` flag;
- `<MASTER_IP>` — the IP address of the cluster's master node.

The installer will then connect to the cluster, retrieve the necessary data, and delete all the resources and objects in the cloud that were created during the DP installation and operation.

Deleting a hybrid cluster

Follow these steps to delete a hybrid cluster consisting of the nodes that were automatically deployed in the cloud as well as the static nodes that were manually plugged in:

1. First, delete all the extra nodes (CloudStatic and Static) that were manually plugged in.
2. Find out the release channel set in the cluster. To do this, run the command:

```
kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
```

3. Run the Deckhouse installer:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
  registry.deckhouse.io/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

where:

- `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
- `<DECKHOUSE_REVISION>` — the Deckhouse edition (e. g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
- `<RELEASE CHANNEL>` — the Deckhouse release channel in kebab-case:
  - `alpha` — for the *Alpha* release channel;
  - `beta` — for the *Beta* release channel;
  - `early-access` — for the *Early Access* release channel;
  - `stable` — for the *Stable* release channel;
  - `rock-solid` — for the *Rock Solid* release channel.

Below is an example command to run the Deckhouse CE installer container:

```
docker run -it --pull=always \
  -v "$PWD/dhctl-tmp:/tmp/dhctl" \
  -v "$HOME/.ssh/:/tmp/.ssh/" registry.deckhouse.io/deckhouse/ce/install:stable bash
```

4. In the container you have started, run the following command:

   ```
   dhctl destroy --ssh-user=<USER> \
     --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
     --yes-i-am-sane-and-i-understand-what-i-am-doing \
     --ssh-host=<MASTER_IP>
   ```

   where:

   - `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to perform actions via `sudo`, add the `--ask-become-pass` flag;
   - `<MASTER IP>` — the IP address of the cluster's master node;
   - `<SSH_PRIVATE KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

The installer will then connect to the cluster, retrieve the necessary data, and delete all the resources and objects in the cloud that were created during the DP installation and operation.

Deleting a static cluster

Follow the steps below to delete a cluster that has been manually installed (e.g., on bare metal):

1. Delete all the extra nodes from the cluster.
2. Find out the release channel set in the cluster. To do this, run the command:

   ```
   kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
   ```

3. Run the Deckhouse installer:

   ```
   docker run --pull=always -it [<MOUNT_OPTIONS>] \
     registry.deckhouse.io/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
   ```

   where:

   - `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
   - `<DECKHOUSE_REVISION>` — the Deckhouse edition (e.g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
   - `<RELEASE CHANNEL>` — the Deckhouse release channel in kebab-case:
     - `alpha` — for the *Alpha* release channel;
     - `beta` — for the *Beta* release channel;
     - `early-access` — for the *Early Access* release channel;
     - `stable` — for the *Stable* release channel;
     - `rock-solid` — for the *Rock Solid* release channel.

   Below is an example command to run the Deckhouse CE installer container:

   ```
   docker run -it --pull=always \
     -v "$PWD/dhctl-tmp:/tmp/dhctl" \
     -v "$HOME/.ssh/:/tmp/.ssh/" registry.deckhouse.io/deckhouse/ce/install:stable bash
   ```

4. Run the command below to delete the cluster:

   ```
   dhctl destroy --ssh-user=<USER> \
     --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
     --yes-i-am-sane-and-i-understand-what-i-am-doing \
     --ssh-host=<MASTER_IP>
   ```

   where:

   - `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to

perform actions via `sudo`, add the `--ask-become-pass` flag;
- `<MASTER IP>` — the IP address of the cluster's master node;
- `<SSH_PRIVATE KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

The installer will then connect to the master node and delete all Deckhouse and Kubernetes cluster components on it.

Additional resources

- Module `deckhouse` parameters
- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Going to production

The following recommendations may be of less importance for a test or development cluster, but they may be critical for a production one.

Release channel and update mode

> Use the `EarlyAccess` or `Stable` release channel. Configure auto-update window or select manual mode.

Select the release channel and update mode that suit your needs. The more stable the release channel is, the later you will have the chance to use the new features.

If possible, use different release channels for clusters. Use a less stable update channel for a development cluster than for a testing cluster or stage (pre-production) cluster.

Use the `EarlyAccess` or `Stable` release channel for production clusters.

If you have more than one cluster in a production environment, make sure that the clusters update at different times. In this case, a new version is installed in one cluster first, and you have time to check it before the other clusters update. To spread the updates over time, set non-overlapping update windows for the clusters or connect them to different release channels. For example, if one cluster gets updates from the `EarlyAccess` channel and another one from the `Stable` channel, a new version reaches the second cluster later.

> ⚠ Even in very busy and critical clusters, it is not a good idea to disable the use of the release channel. The best strategy is a scheduled update. If you are using a Deckhouse Platform release in your cluster that has not received an update in over six months, you will have a hard time getting help quickly should a problem arise.

The update windows management allows you to schedule automatic platform release updates when your cluster is not experiencing peak load.

Kubernetes version

> Use the automatic Kubernetes version selection or set the version explicitly.

In most cases, opt for the automatic selection of the Kubernetes version. In Deckhouse Platform, this behavior is set by default, but it can be changed with the `kubernetesVersion` parameter. Upgrading the Kubernetes version in the cluster has no effect on applications and is done in a consistent and secure fashion.

If the automatic Kubernetes version selection is enabled, the platform can upgrade the Kubernetes version in the cluster together with the platform update (when upgrading a minor version). If the

Kubernetes version in the `kubernetesVersion` parameter is set explicitly, the platform may not upgrade to a newer version at some point if the Kubernetes version used in the cluster is no longer supported.

You must decide for yourself whether to use automatic version selection or set a specific version and update it manually every now and then.

If your application uses outdated versions of resources or depends on a particular version of Kubernetes for some other reason, check whether it is supported and set it explicitly.

Resource requirements

> **i** For production, use a cluster with three master nodes or a cluster with role separation. Choose node resources according to the resource requirements for the selected deployment option and workload type.

The resource requirements for each deployment option are listed on the “System requirements” page. The values in the tables cover only the load of Deckhouse Platform components.

When preparing a production cluster, consider the following:

- Do not use a single-node cluster or a single-master cluster: they are not fault-tolerant.
- For master nodes and monitoring nodes, use fast disks with performance of at least 400 IOPS.
- Set up a dedicated StorageClass on fast disks for Deckhouse Platform components.
- Add resources for the user workload to the resources of worker nodes.
- Run load tests of your applications and adjust node resources based on the results.

For details on choosing resources depending on the load, see the Picking resources for a bare metal cluster guide.

Things to consider when configuring

Master nodes

> **i** Three master nodes with fast 400+ IOPS disks are highly recommended for a cluster.

Use three master nodes in all cases, as they are sufficient for fault tolerance. Also, with three nodes, you can safely update the cluster’s control plane as well as the master nodes. Extra master nodes are not needed, while 2 nodes (or any even number) do not make a quorum.

The master node configuration for cloud clusters can be configured using the masterNodeGroup parameter.

Reference:

- How do I add a master node to a cluster...
- Working with static nodes...

Frontend nodes

> **i** Use two or more frontend nodes.
>
> Use inlet `LoadBalancer` for OpenStack-based clouds and cloud services where automatic balancer ordering is not supported (AWS, GCP, Azure, etc.). Use inlet `HostPort` with an external load balancer for bare metal or vSphere.

Frontend nodes are used for balancing incoming traffic. Such nodes are allocated for Ingress controllers. The NodeGroup of the frontend nodes has a `node-role.deckhouse.io/frontend` label. Read more about allocating nodes for specific load types...

Use more than one frontend node. Frontend nodes must be able to still handle traffic even if one of the frontend nodes fails.

For example, if the cluster has two frontend nodes, each frontend node must be able to handle the entire cluster load in case the second frontend node fails. If the cluster has three frontend nodes, each frontend node must be able to handle a load that is at least one and a half times higher.

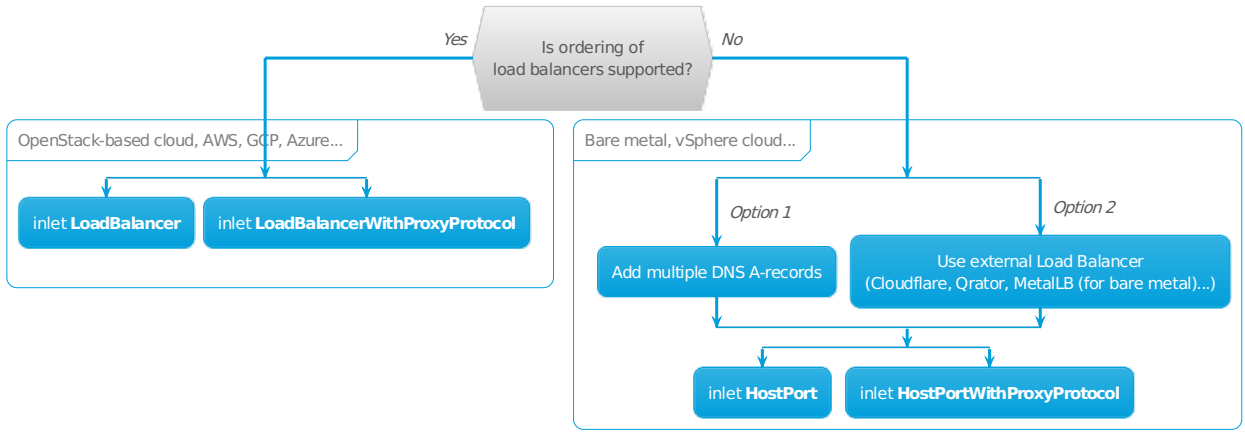
Select the inlet type (it defines the way the traffic comes in).

When deploying the platform in a cloud infrastructure where provisioning of load balancers is supported (e.g., OpenStack-based clouds, AWS, GCP, Azure, etc.), use the `LoadBalancer` or `LoadBalancerWithProxyProtocol` inlet.

In environments where automatic load balancer provisioning is not supported (bare metal clusters, vSphere, custom OpenStack solutions), use the `HostPort` or `HostPortWithProxyProtocol` inlet. In this case, you can either add some A-records to DNS for the corresponding domain or use an external load-balancing service (e.g., Cloudflare, Qrator solutions, or configure metallb).

⚠ The `HostWithFailover` inlet is suitable for clusters with a single frontend node. It reduces the time that the Ingress controller is unavailable during updates. This type of inlet is suitable for important development environments, but **not recommended for production**.

The algorithm for choosing an inlet:



Monitoring nodes

ℹ For high-load clusters, use two monitoring nodes equipped with fast disks.

Monitoring nodes are used to run Grafana, Prometheus, and other monitoring components. The NodeGroup for monitoring nodes has the `node-role.deckhouse.io/monitoring` label attached.

In high-load clusters, where many alerts are generated and many metrics are collected, allocate dedicated nodes for monitoring. If not, monitoring components will be deployed to system nodes.

When allocating monitoring nodes, it is important to allocate fast disks to them. You can do so by providing a dedicated `storageClass` on fast disks for all platform components (global parameter storageClass) or allocate a dedicated `storageClass` to monitoring components only storageClass and longtermStorageClass parameters of the `prometheus` module.

If the cluster is initially created with nodes allocated for a specific type of workload (system nodes, nodes for monitoring, etc.), explicitly specify the corresponding `nodeSelector` parameter in the module configuration for modules that use persistent storage volumes (for example, for the `prometheus` module). For the `prometheus` module, this parameter is nodeSelector.

System nodes

ℹ Dedicate two system nodes.

System nodes are used to run platform modules. Their NodeGroup has the `node-role.deckhouse.io/system` label.

Set two nodes to be system nodes. This way, platform modules will run on them without interfering with user applications in the cluster. Read more about allocating nodes to specific load types….

Provide the platform components with fast disks (the storageClass global parameter).

Cluster subnets

> **i** Plan the cluster subnets before installation, because you cannot change them afterwards.

You set the pod subnet (`podSubnetCIDR`), the service subnet (`serviceSubnetCIDR`), and the node subnet size (`podSubnetNodeCIDRPrefix`) when you install the cluster. Select the pod and service subnets so that they do not overlap with each other or with the node network. Changing these parameters in a running cluster can break it, so the platform blocks such changes. If you need other subnets, recreate the cluster.

The node subnet size defines how many pods fit on a node. By default, the pod limit per node, the `kubelet.maxPods` parameter of the node group, is calculated from this size. The mapping between the subnet size and the pod limit is listed in the Pod limit per node section.

Storage

> **i** Use replicated storage for production data and set a default StorageClass.

Workloads keep persistent data in PersistentVolume volumes, so connect at least one of the supported storage types to the cluster. For production, use storage with data replication between nodes, for example, replicated storage based on DRBD, so the data stays available if a node fails. If you use software-defined storage (SDS), add dedicated disks for data to its nodes.

Set the default StorageClass in the `defaultClusterStorageClass` global parameter. Volumes that do not specify a storage class explicitly get this class.

Virtualization

If virtual machines (VMs) will run in the cluster, follow the recommendations in this section together with the other recommendations of this guide.

Node requirements for VMs

A cluster with VMs must meet the following requirements:

- VMs run on physical servers (bare metal).
- CPUs of nodes for VMs support Intel-VT (VMX) or AMD-V (SVM) instructions, and hardware virtualization is enabled in BIOS or UEFI settings.
- All cluster nodes run the same Linux kernel version.

The same kernel version is necessary for stable VM live migration, because system calls and resource handling differ between kernel versions.

> **⚠** Use virtual machines instead of physical servers as nodes only for demonstration purposes. In this case, enable nested virtualization.

Number of VMs per node

The number of VMs per node is limited by the pod limit, which VMs, containers, and system components share. A running VM with disks usually occupies two pods, because besides the VM pod, a service pod runs on the node and attaches the VM disks and images. It serves disks added with the VirtualMachineBlockDeviceAttachment resource and all disks of a VM with paravirtualization enabled (the `.spec.enableParavirtualization` parameter, enabled by default). During live migration, the same pods start on the target node, so take them into account when you select the node subnet size.

VM subnets

VMs get IP addresses in the main cluster network from the subnets listed in the `virtualMachineCIDRs` parameter of the `virtualization` module. Select these subnets so that they do not overlap with the node network or the cluster subnets, otherwise address conflicts occur.

Specify a subnet by its start address aligned to the mask, for example, `192.168.1.192/27`, not `192.168.1.190/27`. You can specify several subnets and add new ones later, but you cannot delete a subnet while VirtualMachineIPAddress resources hold addresses from it.

The `virtualMachineCIDRs` parameter is optional. Without it, VMs cannot work in the main cluster network or get addresses with the VirtualMachineIPAddress resource, so you can connect them only to

additional networks. VM network configuration is described in the Virtual machine networking section.

Live migration network

By default, live migration traffic goes through the main node network, competes with workloads for bandwidth, and is limited to 640 MiB/s per migration. To keep migrations from affecting workloads and to make them faster, send them to a dedicated network in the `liveMigration.network` parameter of the `virtualization` module, where the platform does not limit the speed by default. This requires the `sdn` module to be enabled and the SystemNetwork resource to be in the `Ready` state. The configuration is described in the Dedicated migration network section.

Running VMs on master nodes

You need to run VMs on master nodes if the cluster has no separate nodes for workloads, for example, in a single-node cluster. To run VMs on master nodes, remove the taints from the `master` node group in the `nodeTemplate.taints` parameter or set tolerations for the selected VMs. Set tolerations in a VM with the `.spec.tolerations` parameter or in a VM class with the `.spec.tolerations` parameter. Removing the taints lets any workload run on master nodes, and tolerations let only the VMs that have them run there.

Storage for VM disks

VM disks and images are stored in PersistentVolume volumes, so select storage for them according to the recommendations in the Storage section. If VM disks will be stored in software-defined storage, take their size into account when you add disks to the nodes of this storage.

Live migration of VMs with local disks is available only in commercial Deckhouse Platform editions, where the disks move together with the VM. In Deckhouse Platform Open, a VM with a local disk cannot move to another node, so you have to stop it during node maintenance.

Image storage

The platform stores VM images in the internal image storage (DVCR), which runs as a single instance on one persistent volume. The storage class and size of this volume are set in the `dvcr.storage` parameter of the `virtualization` module. After the volume is created, you can only increase its size, and you cannot change its storage class, so select the class in advance. Use replicated storage for DVCR so that the images stay available if a node fails.

VM classes

Create your own VM classes (the VirtualMachineClass resource) and do not use the `generic` class for production workloads. The platform creates the `generic` class during installation with the Nehalem CPU model. This model supports the smallest instruction set, so VMs of the `generic` class start on any node but do not get the instructions of modern CPUs.

Mark your main class with the `virtualmachineclass.virtualization.deckhouse.io/is-default-class` annotation set to `true`, and the platform will set it for VMs that do not specify a class. Do not set this annotation on the `generic` class, because it can be lost when the module is updated.

The CPU type of a class is set by the `.spec.cpu.type` parameter and defines the nodes between which VMs can migrate. You cannot change it after you create the class, so select the type in advance:

- `Host` gives a VM an instruction set close to the node CPU. VMs migrate only between nodes with similar CPUs. Migration between Intel and AMD CPUs or between different CPU generations is not possible.
- `HostPassthrough` passes the node CPU to a VM without changes. VMs migrate only to nodes with exactly the same CPU.
- `Discovery` builds a common instruction set for the selected nodes and suits a cluster with different CPUs. The set is fixed when the class is created and does not change when nodes are added to the cluster.

The other CPU types are described in the Virtual processor of virtual machines section.

The `.spec.sizingPolicies` parameter sets the sizing policies of a class. The policies limit the number of cores, the memory size, and the core fraction for VMs of this class. The core fraction is the part of physical core CPU time that the platform guarantees to a VM. The allowed fraction values in the `coreFractions` parameter define how strong CPU oversubscription can be. Oversubscription configuration is described in the CPU oversubscription for virtual machines section.

Access control

> **i** Configure user authentication and grant permissions with the granular role-based model. Create projects for teams.

You can create users and groups in the cluster with the User and Group resources of the `user-authn` module or get them from external authentication providers. The platform supports providers that use OIDC, SAML, and LDAP, as well as GitHub, GitLab, Atlassian Crowd, and Bitbucket Cloud. You can connect several providers to the cluster at the same time. Provider configuration is described in the Integration with external authentication providers section, and creating users in the cluster is described in the Local authentication section.

To isolate team workloads from each other, create projects for the teams (the Project resource of the `multitenancy-manager` module). A project is created from a template that sets resource limits, network isolation, and a security profile. Project management is described in the Project management section.

Grant permissions with the granular role-based model of the `user-authz` module. The model uses standard Kubernetes RBAC resources and the roles that the module creates. Each role acts in one of four scopes:

- **Namespace.** The `d8:namespace:<LEVEL>` roles grant permissions to application users in one namespace and are assigned with a RoleBinding resource.
- **Project.** The `d8:project:<LEVEL>` roles grant permissions to teams in all namespaces of a project at once and are assigned with a ProjectRoleBinding resource.
- **Subsystem.** The `d8:subsystem:<SUBSYSTEM>:<LEVEL>` roles grant permissions to administrators of a part of the platform, for example, the network, and are assigned with a ClusterRoleBinding resource.
- **Entire platform.** The `d8:system:<LEVEL>` roles grant permissions to platform administrators and are assigned with a ClusterRoleBinding resource.

Here, `<LEVEL>` is the access level, for example, `viewer` or `manager`, and `<SUBSYSTEM>` is the subsystem name.

The basic role-based model, built on the ClusterAuthorizationRule and AuthorizationRule resources, still works, but its support will be discontinued in future releases.

Configuring alerts

> **i** You can send alerts using the internal Alertmanager or connect the external one.

Monitoring will work once the platform is installed, however, it is not enough for production clusters. Configure the Alertmanager built in the platform or connect your own Alertmanager to receive incident notifications.

Using the CustomAlertmanager custom resource, you can configure sending alerts to an e-mail, Slack, Telegram, via the webhook, or by other means.

For the list of all available alerts in the Deckhouse Platform monitoring system, refer to the corresponding documentation page.

Collecting logs

> **i** Configure centralized log collection.

Set up centralized log collection from system and user applications using the log-shipper module.

All you have to do is to create a custom resource specifying *what to collect*: ClusterLoggingConfig or PodLoggingConfig; and create a custom resource that specifies where to *send* the collected logs: ClusterLogDestination.

Reference:

- Grafana Loki example
- Logstash example
- Splunk example

Backups

> **i** Set up etcd backups. Have a backup plan ready at all times.

At a minimum, set up etcd backups. This will be your last chance to restore the cluster should things go awry. Keep these backups as *away* from your cluster as possible.

The backups won't help if they don't work or if you don't know how to use them to recover the cluster. Compile a Disaster Recovery Plan (DRP) with specific steps and commands to restore the cluster from a backup. The restore procedures are described in the Backup and restore section.

This plan should be periodically updated and tested in drills.

Community

> **i** Follow the project channel on Telegram for news and updates.

Join the community to keep up with important news and developments. This will help you to share experiences with people who are doing the same thing as you are and avoid typical problems.

Running production in Kubernetes takes effort. Share your experience with Deckhouse Platform in the community to help others switch to Kubernetes.

Additional resources

- Global parameters
- Module `cloud-provider-aws` provider configuration
- Module `control-plane-manager` documentation
- Module `deckhouse` documentation
- Module `ingress-nginx` custom resources
- Module `node-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Picking resources for a bare metal cluster

Before deploying a cluster running Deckhouse Platform, you have to plan the configuration of the future cluster and decide on the parameters of its nodes (e.g., RAM, CPU, etc.).

Installation planning

Before deploying a cluster, you need to plan for the resources that you might need to run the cluster. The following questions will help you plan ahead:

- What is the expected load?
- Does your cluster require a high load mode?
- Does your cluster require a high availability mode?
- Which Deckhouse Platform modules do you intend to use?

The answers to these questions can help you estimate the number of nodes recommended for your cluster deployment. See the Deployment options section of the "System requirements" page to learn more.

> **i** The information below applies to a Deckhouse Platform installation running the Default module set.

Deployment scenarios

Cluster deployment options (single-node cluster, cluster with one master node, cluster with three master nodes or cluster with role separation) and node types are described in the Deployment options

section of the “System requirements” page.

> **i** Starting with version 1.74, Deckhouse Platform has a module integrity control mechanism that protects modules from being replaced or modified. This mechanism is enabled automatically when the operating system on the nodes where Deckhouse Platform is installed supports the `erofs` kernel module. If this kernel module is not present, Deckhouse Platform will continue to operate without the module integrity control mechanism, but an alert will be displayed indicating that this functionality is not working.

Deciding on the amount of resources needed for nodes

Resource requirements depend on the deployment option (single-node cluster, cluster with one master node, cluster with three master nodes or cluster with role separation) and on the workload type (running containers, virtual machines, or both containers and virtual machines).

The tables with node resources for every deployment option and workload type are given in the Resource requirements section of the “System requirements” page. The same section describes what the values in the tables include and which resources to reserve on nodes in addition.

Single-node cluster

> ⚠ Such clusters lack fault tolerance. Do not use this kind of clusters in production environments.

In some cases, a cluster can consist of a single node that performs all the node roles described above. This configuration may be used, for example, to familiarize yourself with the technology or to run workloads with low resource requirements.

The Getting Started guide contains instructions for deploying a single-node cluster. Once you un-taint the node, it will run all cluster components included in the selected module bundle (bundle: Default by default). The resources for such a node are given in the “Single-node cluster” row of the tables in the Resource requirements section for the required workload type: containers, virtual machines or combined workloads.

Such a configuration would allow some workloads to be run.

In a single-node cluster for running containers with the resources from the table, a load of 2500 RPS on a typical web application (e.g., a static Nginx page) consisting of 30 pods, and incoming traffic of 24 Mbps will result in approximately the following resource consumption figures:

- CPU load will increase to ~60% in total;
- RAM and disk resource consumption figures will remain largely unchanged. In the end, however, it comes down to the number of metrics collected and the nature of the workload being run.

> **i** Run load testing of the application and adjust the server capacity according to its results.

Configuration examples

The example below is a cluster with role separation without fault tolerance: one node of each type, without dedicated monitoring nodes. The node resources are taken from the For containers table for the cluster with role separation.

To deploy Deckhouse Platform for running containers in the Ultimate/Enterprise Edition with the Default module set, the following node configuration is required:

- **master nodes** — 1 node, 6 CPU, 12 GB RAM, 60 GB of disk space;
- **frontend nodes** — 1 node, 2 CPU, 4 GB RAM, 60 GB of disk space;
- **system nodes** — 1 node, 6 CPU, 12 GB RAM, 300 GB of disk space (the system node hosts the monitoring components and system PVCs);
- **worker nodes** — 1 node, 2 CPU, 4 GB RAM, 60 GB of disk space plus the resources for the user workload.

> **i** If needed, you can also run Deckhouse Platform in the same configuration on a single node. Its resources are given in the “Single-node cluster” row of the same table: 12 CPU, 24 GB RAM and 350 GB of disk space.

Node and network requirements

The requirements for node hardware, the operating system and the network between nodes are given in the Node requirements and Network requirements sections of the “System requirements” page.

Community

> **i** Join our Telegram channel to stay up to date.

Join the Deckhouse community for updates on important developments and news. There, you will be able to chat with others and learn from their experiences. This way, you can avoid many common mistakes.

Setting up and running a production Kubernetes cluster takes effort. Share your experience with Deckhouse Platform in the community to help others move to Kubernetes.

Additional resources

- Module `deckhouse` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Updating DP in air-gapped environment

> **i** This guide is intended for DP Enterprise Edition, but the mechanism is the same for other editions.
>
> To run the commands given in the guide, you need to authenticate in the container registry `registry.deckhouse.io`. Use `license-token` as the username and your license token as the password for authentication.
>
> The guide uses the third-party utility crane to analyze the container registry. Install it first following the official instructions.

Platform update mechanism using release channels

Deckhouse Platform (DP) updates are based on release channels. You can check which release channel is configured for your DP installation in the `deckhouse` ModuleConfig by running the following command:

```
d8 k get mc deckhouse -o jsonpath='{.spec.settings.releaseChannel}'
```

Example output:

```
Stable
```

Technically, a DP update works as follows: the registry contains an image with the name `release-channel` and a tag corresponding to the release channel. This tag points to an image of a specific DP version (when a new version is released, the tag is updated to point to the new image).

The example below examines the contents of a DP Enterprise Edition image with the Alpha release channel.

To do that, run the following command:

```
crane export registry.deckhouse.io/deckhouse/ee/release-channel:alpha | tar -tf -
```

The output contains a list of files and directories in the image:

```
changelog.yaml
version.json
.werf
.werf/stapel
.werf/tmp
.werf/tmp/ssh-auth-sock
```

The image contains two key files:

- `changelog.yaml`: Describes changes.
- `version.json`: Contains data about the release canary deployment (`canary`), requirements (`requirements`), disruptions (`disruptions`) (deprecated field), and the release version itself in the `version` field.

  To view the contents of `version.json`, run the following command:

  ```
  crane export registry.deckhouse.io/deckhouse/ee/release-channel:alpha | tar -xOf - version.json
  | jq
  ```

  Example contents of `version.json`:

```
{
  "canary": {
    "alpha": {
      "enabled": true,
      "waves": 2,
      "interval": "5m"
    },
    "beta": {
      "enabled": false,
      "waves": 1,
      "interval": "1m"
    },
    "early-access": {
      "enabled": true,
      "waves": 6,
      "interval": "30m"
    },
    "stable": {
      "enabled": true,
      "waves": 6,
      "interval": "30m"
    },
    "rock-solid": {
      "enabled": false,
      "waves": 5,
      "interval": "5m"
    }
  },
  "requirements": {
    "k8s": "1.29",
    "disabledModules": "delivery,l2-load-balancer,ceph-csi",
    "migratedModules": "",
    "autoK8sVersion": "1.31",
    "ingressNginx": "1.9",
    "nodesMinimalOSVersionUbuntu": "18.04",
    "nodesMinimalOSVersionDebian": "10",
    "istioMinimalVersion": "1.21",
    "metallbHasStandardConfiguration": "true",
    "unmetCloudConditions": "true",
    "nodesMinimalLinuxKernelVersion": "5.8.0"
  },
  "disruptions": {
    "1.36": [
      "ingressNginx"
    ]
  },
  "version": "v1.71.5"
}
```


When the `version` field changes in `version.json` in the registry, DP applies a new release in the cluster: a `deckhouserelease` object is created and the update process begins.

> **i** If the `deckhouse` module is set to manual update mode (via the `settings.update.mode` field), you need to manually approve the new version before it is applied.

If there is a gap between the minor version running in the cluster and the one in the `release-channel` image, DP will automatically attempt to reconstruct intermediate `deckhouserelease` objects to perform a sequential update.

> ⚠ Note that DP can’t be updated non-sequentially by skipping minor releases (patch releases are not affected). Minor releases often include migrations that must be applied in order. These migrations may be removed over time. Skipping minor releases may result in:
>
> - Leftover “garbage”
> - Cluster malfunction due to missed migrations

Module from source update mechanism

Modules from source have a similar update mechanism, but their release cycle is decoupled from the platform and fully independent.

The cluster contains ModuleSource resources tracked by DP, which determine the list of available modules.

To see from which repository modules will be installed, run the following command:

```
d8 k get ms deckhouse -o jsonpath='{.spec.registry.repo}'
```

Example output:

```
registry.deckhouse.io/deckhouse/ee/modules
```

You can view repository contents with the following command:

```
crane ls registry.deckhouse.io/deckhouse/ee/modules
```

Example output:

```
commander-agent
console
csi-ceph
csi-nfs
observability
operator-postgres
pod-reloader
prompp
sds-local-volume
sds-node-configurator
sds-replicated-volume
secrets-store-integration
snapshot-controller
stronghold
virtualization
```

As an example, let’s look at the contents of the `console` module image. The registry contains an image with the fixed name `release` and a tag for the channel, pointing to a specific version of the `console` module (when a new module version is released, this tag is updated).

To view the contents of this image, use the following command:

```
crane export registry.deckhouse.io/deckhouse/ee/modules/console/release:alpha | tar -tf -
```

Example output:

```
changelog.yaml
version.json
```

Similarly to the DP image, the module image contains the `changelog.yaml` and `version.json` files.

To view the contents of `version.json`, run the following command:

```
crane export registry.deckhouse.io/deckhouse/ee/modules/console/release:alpha | tar -xOf -
version.json | jq
```

Example contents of `version.json`:

```
{
  "version": "v1.39.4"
}
```

The `version` field contains the module version. When it changes, DP applies a new release (a `modulerelease` object is created and the update process begins).

⚠ Note that modules can't be updated non-sequentially by skipping minor releases (patch releases are not affected). Minor releases often include migrations that must be applied in order. These migrations may be removed over time. Skipping minor releases may result in:

- Leftover "garbage"
- Cluster malfunction due to missed migrations

If required minor versions are missing, DP will report an error that may look as follows: `minor version is greater than deployed $version by one`.

If there is a gap between the minor version running in the cluster and the one in the `release` image, DP will automatically attempt to reconstruct intermediate `modulerelease` objects to perform a sequential update.

Vulnerability scanner database update mechanism

⚠ Available in DP EE and DP Ultimate.

Vulnerability databases are updated every 6 hours. The `operator-trivy` module in the cluster downloads them from the registry once during this period.

Vulnerability database images in DP EE have fixed names and tags and are available at:

```
registry.deckhouse.io/deckhouse/ee/security/trivy-db:2
registry.deckhouse.io/deckhouse/ee/security/trivy-java-db:1
registry.deckhouse.io/deckhouse/ee/security/trivy-checks:0
registry.deckhouse.io/deckhouse/ee/security/trivy-bdu:1
```

To configure periodic updates of vulnerability database images, run the command in the following format:

```
d8 mirror pull --source='registry.deckhouse.io/deckhouse/ee' --
license='<LICENSE_TOKEN>' --no-platform --no-modules $(pwd)/d8-bundle-security-db && d8
mirror push $(pwd)/d8-bundle-security-db <PRIVATE_REGISTRY_HOSTNAME>:5050/dkp/ee --registry-
login='<REGISTRY_LOGIN>' --registry-password='<REGISTRY_PASSWORD>' --tls-skip-verify
```

Here:

- `<LICENSE_TOKEN>`: DP license token.
- `<PRIVATE_REGISTRY_HOSTNAME>`: Address of the private container registry.
- `<REGISTRY_LOGIN>` and `<REGISTRY_PASSWORD>`: Credentials for the private container registry.

Example workflow for updating the platform, modules, and vulnerability databases

To update DP, its modules, and vulnerability databases to the latest versions in an air-gapped environment, download the latest patch releases of all required platform minor versions and modules, then upload them to your registry.

Running `d8 mirror pull --source='registry.deckhouse.io/deckhouse/ee' --license='<LICENSE_TOKEN>' $(pwd)/d8-bundle` downloads all release channel images and all modules (over 30 in DP EE). This results in a very large `d8-bundle` (at the time of writing, the volume

of the `d8-bundle` directory is more than 50 GB).

To avoid this, download only the images relevant to your version following these guidelines:

1. Get the DP version in your cluster using the following command:

   ```
   d8 k -n d8-system get deployment deckhouse -o json | jq -r '.metadata.annotations |
   {"core.deckhouse.io/edition","core.deckhouse.io/version"}'
   ```

   Example output:

   ```
   {
     "core.deckhouse.io/edition": "EE",
     "core.deckhouse.io/version": "v1.68.13"
   }
   ```

2. Get the list of modules deployed in the cluster:

   ```
   d8 k get mr | grep Deployed
   ```

   Example output:

   ```
   commander-agent-v1.2.4             Deployed                    13d
   console-v1.35.1                    Deployed                    7d4h
   ```

   Add this list to the `d8 mirror pull` command as flags: `--include-module='commander-agent@1.2.4' --include-module='console@1.35.1'`

   Alternatively, use a one-liner:

   ```
   d8 k get mr -o json | jq -r '.items[] | select(.status.phase == "Deployed") | "--
   include-module='\''\(.spec.moduleName)@\(.spec.version)'\''"' | paste -sd " " -
   ```

3. Create the final command for pulling images with the obtained parameters:

   ```
   d8 mirror pull --source='registry.deckhouse.io/deckhouse/ee' --
   license='<LICENSE_TOKEN>' --since-version='v1.68.13' --include-
   module='commander-agent@1.2.4' --include-module='console@1.35.1' $(pwd)/d8-bundle
   ```

   > **i** If you have already set up periodic downloading and pushing of vulnerability databases to your registry, you can add the flag `--no-security-db` to skip them during this process.

   This will download the latest patch releases of all required platform minor versions and modules, starting from the latest patch versions up to the current ones on release channels.

4. Push the artifacts to your registry using the following command:

   ```
   d8 mirror push $(pwd)/d8-bundle <PRIVATE_REGISTRY_HOSTNAME>:5050/dkp/ee --registry-
   login='<REGISTRY_LOGIN>' --registry-password='<REGISTRY_PASSWORD>' --tls-skip-
   verify
   ```

5. Check the update status in the cluster by running the following commands:

   ```
   d8 k get deckhousereleases.deckhouse.io
   d8 k get modulereleases.deckhouse.io
   d8 system queue list
   ```

Possible issues

Release is suspended

When trying to download platform images with `d8 mirror pull d8-bundle/ --license='<LICENSE_KEY>'`, you may encounter the following error:

```
Sep  9 00:10:57.145 INFO  ╔ Pull Deckhouse Platform
Sep  9 00:11:01.532 ERROR Pull Deckhouse Platform failed error="Find tags to mirror:
Find versions to mirror: get stable release version from registry: Cannot mirror
Deckhouse: source registry contains suspended release channel \"stable\", try again
later"
Error: pull failed, see the log for details
```

This means that release deployment on a channel has been suspended. This happens when a new version is pushed to the release channel but then something happened that paused its rollout. The channel image is patched with a `suspend` flag.

Nevertheless, you can still download the platform version by specifying the `--deckhouse-tag` flag in `d8 mirror pull`. For example:

```
d8 mirror pull d8-bundle/ --license='<LICENSE_KEY>' --deckhouse-tag='v1.71.3'
```

Example output:

```
Sep 16 12:56:25.074 INFO  ╔ Pull Deckhouse Platform
Sep 16 12:56:25.713 INFO  ╠ Skipped releases lookup as tag "v1.71.3" is specifically
requested with --deckhouse-tag
Sep 16 12:56:25.714 INFO  ╠ Creating OCI Image Layouts
Sep 16 12:56:25.720 INFO  ╠ Resolving tags
Sep 16 12:56:26.715 INFO  ╠╗ Pull release channels and installers
Sep 16 12:56:26.716 INFO  ╠╠ Beginning to pull Deckhouse release channels
information
Sep 16 12:56:26.717 INFO  ╠╠ [1 / 1] Pulling
registry.deckhouse.io/deckhouse/ee/release-channel:v1.71.3
Sep 16 12:56:27.087 INFO  ╠╠ Deckhouse release channels are pulled!
Sep 16 12:56:27.087 INFO  ╠╠ Beginning to pull installers
Sep 16 12:56:27.087 INFO  ╠╠ [1 / 1] Pulling
registry.deckhouse.io/deckhouse/ee/install:v1.71.3
...
```

Additional resources

- Global custom resources
- Module `deckhouse` parameters
- Module `operator-trivy` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Disk layout and size

In the Resource requirements section for various types of Deckhouse Platform nodes, the disk volumes that must be allocated for successful installation and operation of Deckhouse Platform are specified. But you should also pay attention to the correct configuration of the file system so that the disk space does not suddenly “run out” despite the correctly selected volume at the installation stage.

> **i** Problems may arise not through the fault of the administrator, but due to the peculiarities of the installer of the selected Linux distribution. For example, during installation, Astra Linux may allocate 15 GB for the root file system (`/`), 15 GB for the user's home directory (`/home`), and leave the rest unallocated, despite the connected disk having a total capacity of 60 GB, as recommended in the guide. With this configuration, the Deckhouse Platform installation will not fail with an error about insufficient disk space.

To avoid problems in the future, before installing, make sure that the file system partitions of the disk allocated for the machine meet the Deckhouse Platform volume requirements.

Where and what does Deckhouse Platform store

Deckhouse Platform stores various types of data in specific file system directories. The main ones are:

- `/etc/kubernetes/`, `/etc/containerd`, etc. — directories with Kubernetes component configuration;
- `/var/lib/containerd` — layers of images of Deckhouse Platform components and other containers running on the node. The more Deckhouse Platform components are installed on a node or user load containers are launched on it, the more free space in this directory will be required.
- `/var/lib/kubelet` — two types of information are stored in this directory:
  - data about the pods running in the cluster;
  - ephemeral-storage data — for example, if 7 GB of ephemeral-storage is requested on the master node, and there is not enough space in this directory, pods will not be scheduled for this node.
- `/var/lib/etcd` — the etcd database, which stores the information necessary for the operation of the Kubernetes cluster;
- `/var/lib/deckhouse/downloaded/` — repository of release configurations for Deckhouse Platform modules (ModuleRelease);
- `/var/lib/deckhouse/stronghold/` — data storage for Stronghold (if the corresponding module is enabled);
- `/var/log/pods/` — storage of pod logs;
- `/opt/deckhouse/` — Deckhouse Platform service components such as kubelet, containerd, static utilities (e.g. lsblk), etc.;
- `/opt/deckhouse/registry/` (`/var/lib/deckhouse/registry/` when the master NodeGroup has `systemType: DeckhouseEngine`) — the store of the `registry` module, on master nodes only. It holds the in-cluster cache (`storage.cache`) or, in the v1 implementation (Deckhouse Platform up to and including 1.77), the data of the `Proxy` and `Local` modes. When the cache is turned off, the data is kept on purpose (see how to remove leftover cache data). See below for how much space the store needs;
- `/opt/local-path-provisioner/` — directory for storing data when using local storage Local Path Provisioner. The directory is set in the required `spec.path` parameter of LocalPathProvisioner, and the configuration examples use `/opt/local-path-provisioner`.

Disk space recommendations

The sections below show the disk volumes occupied by the various cluster components.

> **i** The total amount of space specified in the tables may exceed the minimum recommended disk size for a cluster node specified in the "Quick Start" or in the Resource requirements section. This is due to the fact that the table shows the maximum required values for the components, and the minimum requirements for the node indicate the average value.

Master nodes

The table below shows the recommended amounts of space for the directories used by Deckhouse Platform on the cluster's master nodes.

Directory	Disk size, GB
`/mnt/vector-data`	1
`/opt` Not counting the `registry` module store: the in-cluster cache or the `Proxy` and `Local` modes of the v1 implementation (details...).	1

Directory	Disk size, GB
`/tmp`	1
`/var/lib` [Show details...]	75
`/var/log/kube-audit`	2
`/var/log/pods`	18 (details...)

Worker nodes

The table below shows the recommended amounts of space for the directories used by Deckhouse Platform on the worker nodes of the cluster.

Directory	Disk size, GB
`/mnt/vector-data`	1
`/opt`	1
`/opt/local-path-provisioner` Depends on the storage settings set by the user. It is recommended to put it on a separate section.	100
`/tmp`	1
`/var/lib` [Show details...]	55
`/var/log/pods`	5 (details...)

System nodes

The system nodes are the nodes on which the Deckhouse Platform components are running. When adding such nodes to the cluster, keep in mind that they host the monitoring load, including:

- Prometheus;
- loki;
- upmeter and other Deckhouse Platform components.

If monitoring data is stored locally on the nodes, it is recommended to additionally allocate ≥ 100 GB of free disk space for each system node.

> **i** If you use a cluster configuration without dedicated system nodes, the above load will be distributed to other nodes, and you need to take into account the recommended amount of disk storage when choosing their configuration.

Pod log storage

Pod logs are stored in the `/var/log/pods/` directory. The amount of logs used depends on the number of containers and Deckhouse Platform settings. On average, about 90 containers are running on the master node when using the Default module set, By default, up to 4 log files of up to 50 MB each are stored for each container, that is, up to 200 MB per container. That is, there should be a minimum of `90 * 200 MB = 18 GB` space available in the directory `/var/log/pods/`.

The log storage parameters can be redefined in the `spec.kubelet.containerLogMaxSize` and `spec.kubelet.containerLogMaxFiles` parameters of the NodeGroup (see node customization). Default values:

```
spec:
  kubelet:
    containerLogMaxSize: 50Mi
    containerLogMaxFiles: 4
```

Trivy vulnerability database repository

Deckhouse Platform has a built-in vulnerability image scanning system based on Trivy, which scans all

container images used in the cluster's pods. Both public vulnerability databases and enriched data from Astra Linux, ALT Linux and RED OS are used for scanning. The total amount of disk space occupied by databases is 5 GB, so it must be taken into account when choosing the disk partition configuration.

Databases are stored on the cluster's system nodes, and if there are no such nodes in the cluster, the databases will be located on the worker node.

Registry store on master nodes

The master node table above does not include the store of the in-cluster cache of the `registry` module. If the cache is enabled (`storage.cache`), every master node keeps its own copy of the store in `/opt/deckhouse/registry` (`/var/lib/deckhouse/registry` when the master NodeGroup has `systemType: DeckhouseEngine`), and the file system that holds this directory needs extra space:

- for the store budget set in `storage.size`: plan for three releases of the image set the cluster uses (the deployed one, the previous one and the next one) together with its modules and the module catalog;
- for the reserve that the store always leaves free for the node on that file system. Because of the reserve, the file system must be noticeably larger than `storage.size`, including when the store has a disk of its own.

It is recommended to mount a separate disk at the store directory: if the store shares a disk with etcd, etcd slows down while the cache is being filled. Without a separate disk, reserve the space on the file system that holds the store directory (with a DeckhouseEngine master NodeGroup, the one that holds `/var/lib/deckhouse`). If `/opt` is a separate partition sized by the table above (1 GB), the store does not fit there.

The `Proxy` and `Local` modes of the v1 implementation also keep their data in `/opt/deckhouse/registry`.

For sizing details, see "Requirements for using the module" in the `registry` module documentation.

If resource limits are configured

If resource limits in terms of disk space are configured for cluster entities, then the necessary free disk space on the node must be available in any case, otherwise the load will be displaced from these nodes.

LVM-based local storage

In a Deckhouse Platform cluster, you can configure local storage on nodes, using LVM.

Requirements and placement procedure:

- Free block devices (disk partitions) should be available on the node.
- These devices will be used by the sds-local-volume module to create a StorageClass.
- The amount of free space on the block device must correspond to the amount that is planned to be provided through the created StorageClass.

Additional resources

- Global custom resources
- Module `loki` documentation
- Module `node-manager` custom resources
- Module `prometheus` documentation
- Module `registry` documentation
- Module `upmeter` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Installing Deckhouse Platform in a private environment

This guide describes how to deploy a Deckhouse Platform cluster in a private environment with no direct access to the Deckhouse Platform container image registry (`registry.deckhouse.io`) and to external deb/rpm package repositories used on nodes running supported operating systems.

Private environment specifics

Deploying in a private environment is almost the same as deploying on bare metal.

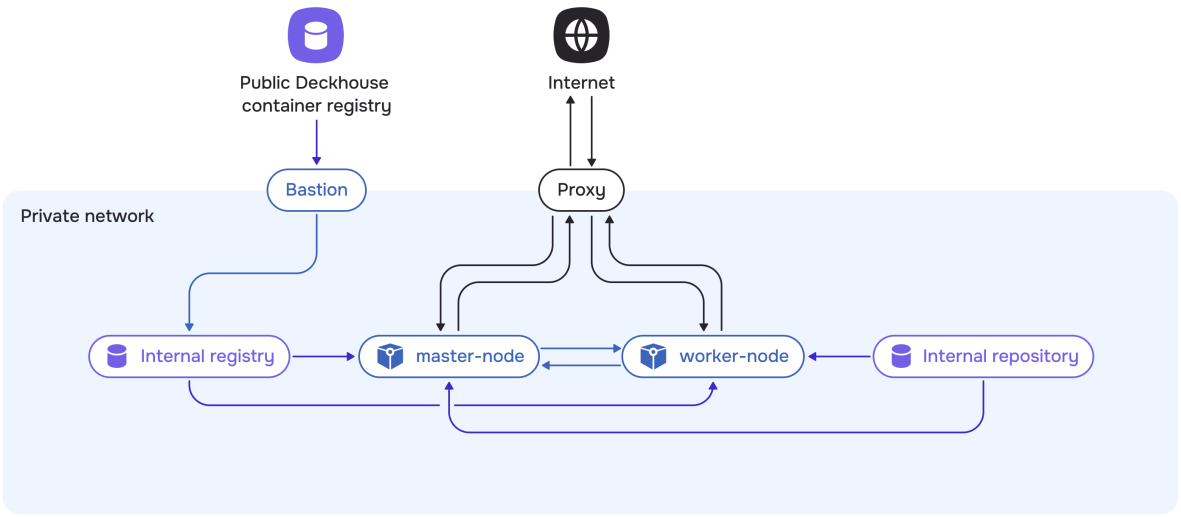
Key specifics:

- Proxy server parameters set in the cluster configuration during installation are automatically propagated to the `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables for **cluster nodes and Deckhouse Platform components**. User applications (pods) do not receive these variables from the cluster configuration automatically. To give them Internet access via a proxy, you must set the environment variables (`HTTP_PROXY`, `HTTPS_PROXY`, and, if needed, `NO_PROXY`) explicitly in manifests. Depending on corporate policy, application access may be arranged in other ways—for example, by allowing direct egress from nodes.
- A container registry with Deckhouse Platform images is deployed separately with access from inside the perimeter, and the cluster is configured to use it with the required permissions.

Cluster nodes are usually accessed through a dedicated physical server or virtual machine called a bastion host. A proxy for access to external resources from the internal network is deployed according to your network policy and infrastructure architecture; depending on requirements, it may run on the bastion host or on a separate machine. A private container registry should preferably run on a separate VM or server in the internal network. Colocating the registry on the bastion host is not recommended for production. An exception may be lab or simplified stand-alone setups for limited use cases.

> **i** Depending on your organization’s security policies, access to external resources may be fully disabled. In that case, no proxy is used for outbound access to external networks. Required external artifacts (for example, an archive of Deckhouse Platform container images) are brought into the perimeter onto the target VM by any permitted means—for example, removable media.

Overall private environment diagram:



> **i** The diagram also shows an internal OS package repository. It is used to install packages on nodes when access to official repositories is not available even through a proxy. Many private environments already run internal OS package mirrors, and installation uses them—in that case, a proxy for package traffic is not required. A proxy server is used for other kinds of traffic:
>
> - pulling container images from the public Deckhouse Platform registry to the bastion host
> - Deckhouse Platform components and nodes calling external resources (if allowed by policy)
> - optionally, pod applications accessing external services.

Infrastructure selection

This guide describes deploying a cluster in a private environment consisting of one master node and one worker node.

You will need:

- a personal computer from which you run operations
- a dedicated physical server or virtual machine for the bastion host
- a dedicated physical server or virtual machine for the container registry
- optionally, a physical server or virtual machine for the proxy server
- two physical servers or two virtual machines for the cluster nodes.

Server requirements:

- **Bastion**: at least 4 CPU cores, 8 GB RAM, and 150 GB on fast storage. That much disk space is needed because the bastion host temporarily holds all Deckhouse Platform images used for installation. Images are downloaded from the public Deckhouse Platform registry to the bastion host before being pushed to the private container registry and packed into archives; these steps require substantial free space.
- **VM for the private registry**: at least 4 CPU cores, 8 GB RAM, and at least 150 GB on fast storage for Deckhouse Platform images. Plan disk capacity with a margin, using the bundle size after `d8 mirror push` as a guide.
- **Cluster nodes**: choose resources for future cluster nodes based on expected workload. For example, the minimum recommended configuration is 4 CPU cores (*8 CPU cores recommended*), 8 GB RAM (*16 GB RAM recommended*), and 60 GB on fast storage (400+ IOPS) per node. If you plan to enable the in-cluster cache of the `registry` module, the master nodes also need disk space for the cache store (see “Disk layout and size”).

Preparing a private container registry

> ⚠ Deckhouse Platform supports Basic and Bearer token authentication schemes for container registries (Basic is tried first; if it fails, Bearer is used).
>
> If a reverse proxy is placed in front of the registry, it must correctly forward the Registry API v2 header `Docker-Distribution-API-Version: registry/2.0`. Otherwise the Basic check may fail, and the subsequent Bearer attempt may fail with the error `couldn't find bearer realm parameter`.

You may use any supported private container registry. Compatibility has been tested and is guaranteed for the following: Nexus, Harbor, Artifactory, Docker Registry, and Quay.

The instructions below cover two private container registry options — Docker Registry and Harbor. Choose the matching tab.

Installing a container registry

- Docker Registry
- Harbor

Install Docker Registry from your distribution’s repositories. For example, on a RHEL-based OS:

```
dnf install registry httpd-tools
```

Create a user and password with htpasswd. For example, to create user `deckhouse` with password `<PASSWORD>`:

```
htpasswd -bcB /etc/registry/htpasswd deckhouse <PASSWORD>
```

Generate a self-signed certificate for the registry:

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout
/etc/registry/private.key -out /etc/registry/public.crt -days 3650 -nodes -subj
"/C=US/ST=California/L=SanFrancisco/O=example/OU=IT/CN=registry.local" -addext
"subjectAltName=DNS:registry.local"
```

Replace `registry.local` with the address that will be used for the image registry in your network.

Update the registry configuration with the certificates you created. Edit `/etc/registry/config.yml`:

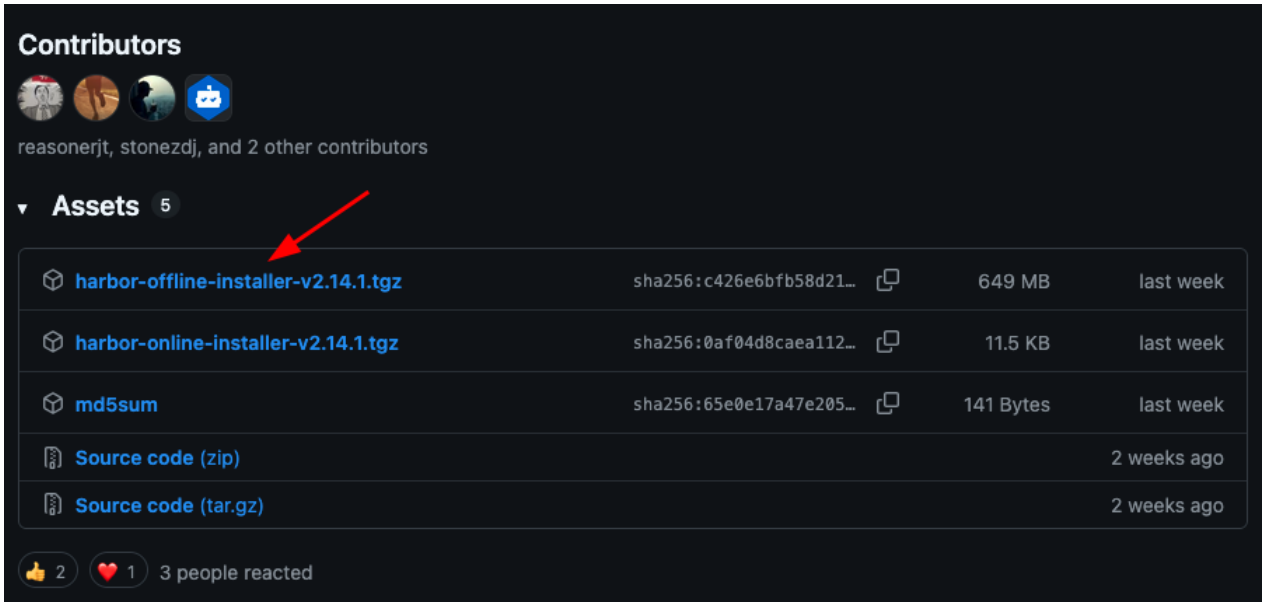
```
version: 0.1
log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
  delete:
    enabled: true
http:
  addr: :443
  tls:
    certificate: /etc/registry/public.crt
    key: /etc/registry/private.key
  headers:
    X-Content-Type-Options: [nosniff]
auth:
  htpasswd:
    realm: basic-realm
    path: /etc/registry/htpasswd
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
```

Enable the service at boot and confirm it is running:

```
systemctl enable --now registry.service
systemctl status registry.service
```

Later sections of this guide use Harbor as the example. It supports policy configuration and role-based access control (RBAC), vulnerability scanning, and marking trusted artifacts. Harbor is a CNCF project.

Install the latest Harbor release from the project's GitHub releases page. Download the installer archive from the desired release, selecting the asset with `harbor-offline-installer` in its name.



Copy the download URL. For example, for `harbor-offline-installer-v2.14.1.tgz` it will look like this: `https://github.com/goharbor/harbor/releases/download/v2.14.1/harbor-offline-installer-v2.14.1.tgz`.

Connect via SSH to the **virtual machine where Harbor will run** and download the archive using any convenient method. If that VM has no direct Internet access, download the archive on your workstation

or on the bastion host, then copy it to the Harbor VM.

How to download the archive with wget...

Run the command (use the current URL):

```
wget https://github.com/goharbor/harbor/releases/download/v2.14.1/harbor-offline-installer-v2.14.1.tgz
```

How to download the archive with curl...

Run the command (use the current URL):

```
curl -O https://github.com/goharbor/harbor/releases/download/v2.14.1/harbor-offline-installer-v2.14.1.tgz
```

Extract the downloaded archive (specify the archive name):

```
tar -zxf ./harbor-offline-installer-v2.14.1.tgz
```

The extracted `harbor` directory contains the files required for installation.

On **the same VM**, install Docker and the Docker Compose plugin. You will need them to configure TLS access to the registry and to run the Harbor installer.

Before deploying the registry, generate a self-signed TLS certificate.

> **i** Due to access restrictions in a private environment, it is not possible to obtain certificates from services such as Let's Encrypt, since the service will not be able to perform the validation required to issue a certificate.
>
> There are several ways to generate certificates. This guide describes one of them. If needed, use any other suitable approach or provide an existing certificate.

Create the `certs` directory inside the `harbor` directory:

```
cd harbor/
mkdir certs
cd certs
```

Generate a self-signed certificate for Harbor:

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout registry.local.key -out registry.local.crt -sha256 -days 3650 -nodes -subj "/C=US/ST=California/L=SanFrancisco/O=example/OU=IT/CN=registry.local" -addext "subjectAltName=DNS:registry.local"
```

Replace `registry.local` with the address that will be used for the image registry in your network.

Verify that the certificates were created successfully:

```
ls -la
```

Example command output...

```
$ ls -la
total 12
drwxrwxr-x 2 ubuntu ubuntu 4096 Dec  5 14:58 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Dec  4 12:53 ..
-rw-rw-r-- 1 ubuntu ubuntu  909 Dec  5 14:58 registry.local.crt
-rw------- 1 ubuntu ubuntu  292 Dec  5 14:57 registry.local.key
```

Next, configure Docker to work with the private container registry over TLS. Create the `registry.local` directory under `/etc/docker/certs.d/`:

```
sudo mkdir -p /etc/docker/certs.d/registry.local
```

> **i** The `-p` option tells `mkdir` to create parent directories if they do not exist (in this case, the `certs.d` directory).

Copy the certificate into it as the root CA (for a self-signed certificate this is the same file):

```
cp registry.local.crt /etc/docker/certs.d/registry.local/ca.crt
```

This certificate will be used when accessing the registry via the `registry.local` domain name.

Return to the `harbor` directory (installer root):

```
cd ..
```

Copy the configuration file template that comes with the installer:

```
cp harbor.yml.tmpl harbor.yml
```

Update the following parameters in `harbor.yml`:

- `hostname`: set to `registry.local` (the certificates were generated for this name)
- `certificate`: specify the path to the generated certificate in the `certs` directory (for example, `/home/ubuntu/harbor/certs/registry.local.crt`)
- `private_key`: specify the path to the private key (for example, `/home/ubuntu/harbor/certs/registry.local.key`)
- `harbor_admin_password`: set a password for accessing the web UI.

Save the file.

Example configuration file...

```
# Configuration file of Harbor

# The IP address or hostname to access admin UI and registry service.
# DO NOT use localhost or 127.0.0.1, because Harbor needs to be accessed by external
clients.
hostname: registry.local

# http related config
http:
  # port for http, default is 80. If https enabled, this port will redirect to https
port
  port: 80

# https related config
https:
  # https port for harbor, default is 443
  port: 443
  # The path of cert and key files for nginx
  certificate: /home/ubuntu/harbor/certs/registry.local.crt
  private_key: /home/ubuntu/harbor/certs/registry.local.key
  # enable strong ssl ciphers (default: false)
  # strong_ssl_ciphers: false

# # Harbor will set ipv4 enabled only by default if this block is not configured
# # Otherwise, please uncomment this block to configure your own ip_family stacks
# ip_family:
#   # ipv6Enabled set to true if ipv6 is enabled in docker network, currently it
affected the nginx related component
#   ipv6:
#     enabled: false
#   # ipv4Enabled set to true by default, currently it affected the nginx related
```

```
component
#   ipv4:
#     enabled: true

# # Uncomment following will enable tls communication between all harbor components
# internal_tls:
#   # set enabled to true means internal tls is enabled
#   enabled: true
#   # put your cert and key files on dir
#   dir: /etc/harbor/tls/internal


# Uncomment external_url if you want to enable external proxy
# And when it enabled the hostname will no longer used
# external_url: https://reg.mydomain.com:8433

# The initial password of Harbor admin
# It only works in first time to install harbor
# Remember Change the admin password from UI after launching Harbor.
harbor_admin_password: <PASSWORD>

# Harbor DB configuration
database:
  # The password for the user('postgres' by default) of Harbor DB. Change this
before any production use.
  password: <PASSWORD>
  # The maximum number of connections in the idle connection pool. If it <=0, no
idle connections are retained.
  max_idle_conns: 100
  # The maximum number of open connections to the database. If it <= 0, then there
is no limit on the number of open connections.
  # Note: the default number of connections is 1024 for postgres of harbor.
  max_open_conns: 900
  # The maximum amount of time a connection may be reused. Expired connections may
be closed lazily before reuse. If it <= 0, connections are not closed due to a
connection's age.
  # The value is a duration string. A duration string is a possibly signed sequence
of decimal numbers, each with optional fraction and a unit suffix, such as "300ms",
"-1.5h" or "2h45m". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h".
  conn_max_lifetime: 5m
  # The maximum amount of time a connection may be idle. Expired connections may be
closed lazily before reuse. If it <= 0, connections are not closed due to a
connection's idle time.
  # The value is a duration string. A duration string is a possibly signed sequence
of decimal numbers, each with optional fraction and a unit suffix, such as "300ms",
"-1.5h" or "2h45m". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h".
  conn_max_idle_time: 0

# The default data volume
data_volume: /data

# Harbor Storage settings by default is using /data dir on local filesystem
# Uncomment storage_service setting If you want to using external storage
# storage_service:
#   # ca_bundle is the path to the custom root ca certificate, which will be
injected into the truststore
#   # of registry's containers.  This is usually needed when the user hosts a
internal storage with self signed certificate.
#   ca_bundle:

#   # storage backend, default is filesystem, options include filesystem, azure,
gcs, s3, swift and oss
#   # for more info about this configuration please refer
https://distribution.github.io/distribution/about/configuration/
```

```
#   # and https://distribution.github.io/distribution/storage-drivers/
#   filesystem:
#     maxthreads: 100
#   # set disable to true when you want to disable registry redirect
#   redirect:
#     disable: false

# Trivy configuration
#
# Trivy DB contains vulnerability information from NVD, Red Hat, and many other
upstream vulnerability databases.
# It is downloaded by Trivy from the GitHub release page
https://github.com/aquasecurity/trivy-db/releases and cached
# in the local file system. In addition, the database contains the update timestamp
so Trivy can detect whether it
# should download a newer version from the Internet or use the cached one.
Currently, the database is updated every
# 12 hours and published as a new release to GitHub.
trivy:
  # ignoreUnfixed The flag to display only fixed vulnerabilities
  ignore_unfixed: false
  # skipUpdate The flag to enable or disable Trivy DB downloads from GitHub
  #
  # You might want to enable this flag in test or CI/CD environments to avoid GitHub
rate limiting issues.
  # If the flag is enabled you have to download the `trivy-offline.tar.gz` archive
manually, extract `trivy.db` and
  # `metadata.json` files and mount them in the `/home/scanner/.cache/trivy/db`
path.
  skip_update: false
  #
  # skipJavaDBUpdate If the flag is enabled you have to manually download the
`trivy-java.db` file and mount it in the
  # `/home/scanner/.cache/trivy/java-db/trivy-java.db` path
  skip_java_db_update: false
  #
  # The offline_scan option prevents Trivy from sending API requests to identify
dependencies.
  # Scanning JAR files and pom.xml may require Internet access for better detection,
but this option tries to avoid it.
  # For example, the offline mode will not try to resolve transitive dependencies in
pom.xml when the dependency doesn't
  # exist in the local repositories. It means a number of detected vulnerabilities
might be fewer in offline mode.
  # It would work if all the dependencies are in local.
  # This option doesn't affect DB download. You need to specify "skip-update" as
well as "offline-scan" in an air-gapped environment.
  offline_scan: false
  #
  # Comma-separated list of what security issues to detect. Possible values are
`vuln`, `config` and `secret`. Defaults to `vuln`.
  security_check: vuln
  #
  # insecure The flag to skip verifying registry certificate
  insecure: false
  #
  # timeout The duration to wait for scan completion.
  # There is upper bound of 30 minutes defined in scan job. So if this `timeout` is
larger than 30m0s, it will also timeout at 30m0s.
  timeout: 5m0s
  #
  # github_token The GitHub access token to download Trivy DB
```

```
  #
  # Anonymous downloads from GitHub are subject to the limit of 60 requests per
hour. Normally such rate limit is enough
  # for production operations. If, for any reason, it's not enough, you could
increase the rate limit to 5000
  # requests per hour by specifying the GitHub access token. For more details on
GitHub rate limiting please consult
  # https://docs.github.com/rest/overview/resources-in-the-rest-api#rate-limiting
  #
  # You can create a GitHub token by following the instructions in
  # https://help.github.com/en/github/authenticating-to-github/creating-a-personal-
access-token-for-the-command-line
  #
  # github_token: xxx

jobservice:
  # Maximum number of job workers in job service
  max_job_workers: 10
  # Maximum hours of task duration in job service, default 24
  max_job_duration_hours: 24
  # The jobLoggers backend name, only support "STD_OUTPUT", "FILE" and/or "DB"
  job_loggers:
    - STD_OUTPUT
    - FILE
    # - DB
  # The jobLogger sweeper duration (ignored if `jobLogger` is `stdout`)
  logger_sweeper_duration: 1 #days

notification:
  # Maximum retry count for webhook job
  webhook_job_max_retry: 3
  # HTTP client timeout for webhook job
  webhook_job_http_client_timeout: 3 #seconds

# Log configurations
log:
  # options are debug, info, warning, error, fatal
  level: info
  # configs for logs in local storage
  local:
    # Log files are rotated log_rotate_count times before being removed. If count is
0, old versions are removed rather than rotated.
    rotate_count: 50
    # Log files are rotated only if they grow bigger than log_rotate_size bytes. If
size is followed by k, the size is assumed to be in kilobytes.
    # If the M is used, the size is in megabytes, and if G is used, the size is in
gigabytes. So size 100, size 100k, size 100M and size 100G
    # are all valid.
    rotate_size: 200M
    # The directory on your host that store log
    location: /var/log/harbor

  # Uncomment following lines to enable external syslog endpoint.
  # external_endpoint:
  #   # protocol used to transmit log to external endpoint, options is tcp or udp
  #   protocol: tcp
  #   # The host of external endpoint
  #   host: localhost
  #   # Port of external endpoint
  #   port: 5140

#This attribute is for migrator to detect the version of the .cfg file, DO NOT
MODIFY!
```

```
_version: 2.14.0

# Uncomment external_database if using external database.
# external_database:
#   harbor:
#     host: harbor_db_host
#     port: harbor_db_port
#     db_name: harbor_db_name
#     username: harbor_db_username
#     password: <HARBOR_DB_PASSWORD>
#     ssl_mode: disable
#     max_idle_conns: 2
#     max_open_conns: 0

# Uncomment redis if need to customize redis db
# redis:
#   # db_index 0 is for core, it's unchangeable
#   # registry_db_index: 1
#   # jobservice_db_index: 2
#   # trivy_db_index: 5
#   # it's optional, the db for harbor business misc, by default is 0, uncomment it
if you want to change it.
#   # harbor_db_index: 6
#   # it's optional, the db for harbor cache layer, by default is 0, uncomment it if
you want to change it.
#   # cache_layer_db_index: 7

# Uncomment external_redis if using external Redis server
# external_redis:
#   # support redis, redis+sentinel
#   # host for redis: <host_redis>:<port_redis>
#   # host for redis+sentinel:
#   #  <host_sentinel1>:<port_sentinel1>,<host_sentinel2>:<port_sentinel2>,
<host_sentinel3>:<port_sentinel3>
#   host: redis:6379
#   password:
#   # Redis AUTH command was extended in Redis 6, it is possible to use it in the
two-arguments AUTH <username> <password> form.
#   # there's a known issue when using external redis username
ref:https://github.com/goharbor/harbor/issues/18892
#   # if you care about the image pull/push performance, please refer to this
https://github.com/goharbor/harbor/wiki/Harbor-FAQs#external-redis-username-
password-usage
#   # username:
#   # sentinel_master_set must be set to support redis+sentinel
#   #sentinel_master_set:
#   # tls configuration for redis connection
#   # only server-authentication is supported
#   # mtls for redis connection is not supported
#   # tls connection will be disable by default
#   tlsOptions:
#     enable: false
#   # if it is a self-signed ca, please set the ca path specifically.
#     rootCA:
#   # db_index 0 is for core, it's unchangeable
#   registry_db_index: 1
#   jobservice_db_index: 2
#   trivy_db_index: 5
#   idle_timeout_seconds: 30
#   # it's optional, the db for harbor business misc, by default is 0, uncomment it
if you want to change it.
#   # harbor_db_index: 6
#   # it's optional, the db for harbor cache layer, by default is 0, uncomment it if
```

```
you want to change it.
#   # cache_layer_db_index: 7

# Uncomment uaa for trusting the certificate of uaa instance that is hosted via
self-signed cert.
# uaa:
#   ca_file: /path/to/ca

# Global proxy
# Config http proxy for components, e.g. http://my.proxy.com:3128
# Components doesn't need to connect to each others via http proxy.
# Remove component from `components` array if want disable proxy
# for it. If you want use proxy for replication, MUST enable proxy
# for core and jobservice, and set `http_proxy` and `https_proxy`.
# Add domain to the `no_proxy` field, when you want disable proxy
# for some special registry.
proxy:
  http_proxy:
  https_proxy:
  no_proxy:
  components:
    - core
    - jobservice
    - trivy

# metric:
#   enabled: false
#   port: 9090
#   path: /metrics

# Trace related config
# only can enable one trace provider(jaeger or otel) at the same time,
# and when using jaeger as provider, can only enable it with agent mode or collector
mode.
# if using jaeger collector mode, uncomment endpoint and uncomment username,
password if needed
# if using jaeger agetn mode uncomment agent_host and agent_port
# trace:
#   enabled: true
#   # set sample_rate to 1 if you wanna sampling 100% of trace data; set 0.5 if you
wanna sampling 50% of trace data, and so forth
#   sample_rate: 1
#   # # namespace used to differentiate different harbor services
#   # namespace:
#   # # attributes is a key value dict contains user defined attributes used to
initialize trace provider
#   # attributes:
#   #   application: harbor
#   # # jaeger should be 1.26 or newer.
#   # jaeger:
#   #   endpoint: http://hostname:14268/api/traces
#   #   username:
#   #   password:
#   #   agent_host: hostname
#   #   # export trace data by jaeger.thrift in compact mode
#   #   agent_port: 6831
#   # otel:
#   #   endpoint: hostname:4318
#   #   url_path: /v1/traces
#   #   compression: false
#   #   insecure: true
#   #   # timeout is in seconds
#   #   timeout: 10
```

```
# Enable purge _upload directories
upload_purging:
  enabled: true
  # remove files in _upload directories which exist for a period of time, default is
one week.
  age: 168h
  # the interval of the purge operations
  interval: 24h
  dryrun: false

# Cache layer configurations
# If this feature enabled, harbor will cache the resource
# `project/project_metadata/repository/artifact/manifest` in the redis
# which can especially help to improve the performance of high concurrent
# manifest pulling.
# NOTICE
# If you are deploying Harbor in HA mode, make sure that all the harbor
# instances have the same behaviour, all with caching enabled or disabled,
# otherwise it can lead to potential data inconsistency.
cache:
  # not enabled by default
  enabled: false
  # keep cache for one day by default
  expire_hours: 24

# Harbor core configurations
# Uncomment to enable the following harbor core related configuration items.
# core:
#   # The provider for updating project quota(usage), there are 2 options, redis or
db,
#   # by default is implemented by db but you can switch the updation via redis
which
#   # can improve the performance of high concurrent pushing to the same project,
#   # and reduce the database connections spike and occupies.
#   # By redis will bring up some delay for quota usage updation for display, so
only
#   # suggest switch provider to redis if you were ran into the db connections spike
around
#   # the scenario of high concurrent pushing to same project, no improvement for
other scenes.
#   quota_update_provider: redis # Or db
```

Run the installation script:

```
./install.sh
```

Harbor installation will start: the required images will be prepared and the containers will be started.

Successful installation log...

```
...
[Step 5]: starting Harbor ...
[+] up 10/10
 ✔ Network harbor_harbor       Created 0.0s
 ✔ Container harbor-log        Created 0.1s
 ✔ Container registry          Created 0.1s
 ✔ Container harbor-portal     Created 0.2s
 ✔ Container redis             Created 0.1s
 ✔ Container harbor-db         Created 0.1s
 ✔ Container registryctl       Created 0.2s
 ✔ Container harbor-core       Created 0.1s
 ✔ Container nginx             Created 0.1s
 ✔ Container harbor-jobservice Created 0.1s
✔ ----Harbor has been installed and started successfully.----
```

Verify that Harbor is running successfully:

```
docker ps
```

Example command output...

```
CONTAINER ID   IMAGE                                  COMMAND
CREATED         STATUS                   PORTS
NAMES
df1636bd1295   goharbor/nginx-photon:v2.14.1          "nginx -g 'daemon of…"   3
minutes ago   Up 3 minutes (healthy)   0.0.0.0:80->8080/tcp, [::]:80->8080/tcp, 0.0.0.0:443-
>8443/tcp, [::]:443->8443/tcp   nginx
15fe1abdf9b1   goharbor/harbor-jobservice:v2.14.1     "/harbor/entrypoint.…"   3
minutes ago   Up 3 minutes (healthy)
harbor-jobservice
9b006f03821e   goharbor/harbor-core:v2.14.1           "/harbor/entrypoint.…"   3
minutes ago   Up 3 minutes (healthy)
harbor-core
fbd35346573e   goharbor/registry-photon:v2.14.1       "/home/harbor/entryp…"   3
minutes ago   Up 3 minutes (healthy)
registry
c199a232fdb6   goharbor/harbor-registryctl:v2.14.1    "/home/harbor/start.…"   3
minutes ago   Up 3 minutes (healthy)
registryctl
a78d9a1a5b0b   goharbor/harbor-db:v2.14.1             "/docker-entrypoint.…"   3
minutes ago   Up 3 minutes (healthy)
harbor-db
89d6c922b78a   goharbor/harbor-portal:v2.14.1         "nginx -g 'daemon of…"   3
minutes ago   Up 3 minutes (healthy)
harbor-portal
ef18d7f24777   goharbor/redis-photon:v2.14.1          "redis-server /etc/r…"   3
minutes ago   Up 3 minutes (healthy)
redis
9330bcce48be   goharbor/harbor-log:v2.14.1            "/bin/sh -c /usr/loc…"   3
minutes ago   Up 3 minutes (healthy)   127.0.0.1:1514->10514/tcp
harbor-log
```

On the Harbor VM, add an entry to `/etc/hosts` that maps the `registry.local` domain name to `localhost` so you can open Harbor by that name from the same machine:

```
127.0.0.1 localhost registry.local
```

⚠ In some cloud providers (for example, Yandex Cloud), changes to `/etc/hosts` may be reverted after a virtual machine reboot. A note about this is typically shown at the beginning of the `/etc/hosts` file.

```
# Your system has configured 'manage_etc_hosts' as True.
# As a result, if you wish for changes to this file to persist
# then you will need to either
# a.) make changes to the master file in /etc/cloud/templates/hosts.debian.tmpl
# b.) change or remove the value of 'manage_etc_hosts' in
#     /etc/cloud/cloud.cfg or cloud-config from user-data
```

If your provider uses the same mechanism, apply the corresponding changes to the template file referenced in the comment so that the settings persist after reboot.

Harbor installation is now complete.

Configuring Harbor

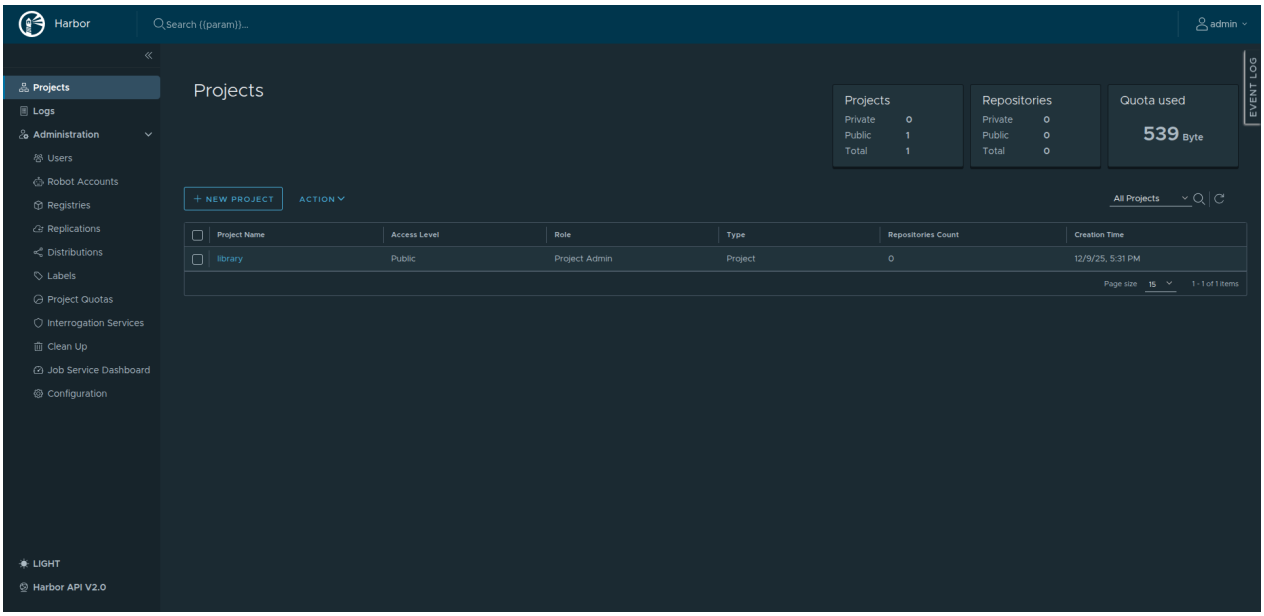
Create a project and credentials used to work with it.

Open the Harbor web UI at `registry.local`. Access to this UI from the public Internet is intentionally blocked; connect only from a host that has access to the internal network.



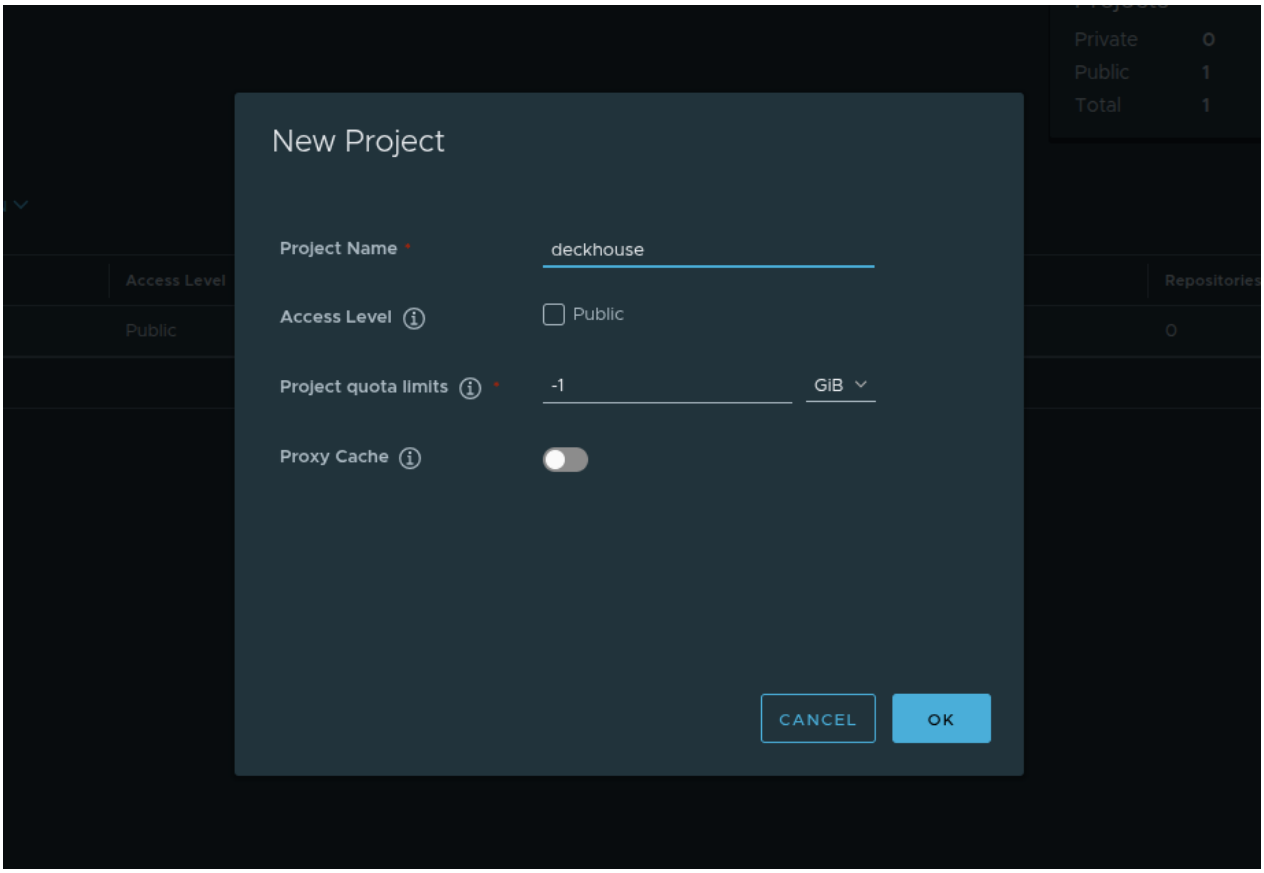
ℹ To open Harbor by the `registry.local` domain name from your workstation, add a matching entry to `/etc/hosts` pointing to the Harbor VM's internal IP address.

Sign in with the username and password from `harbor.yml`.



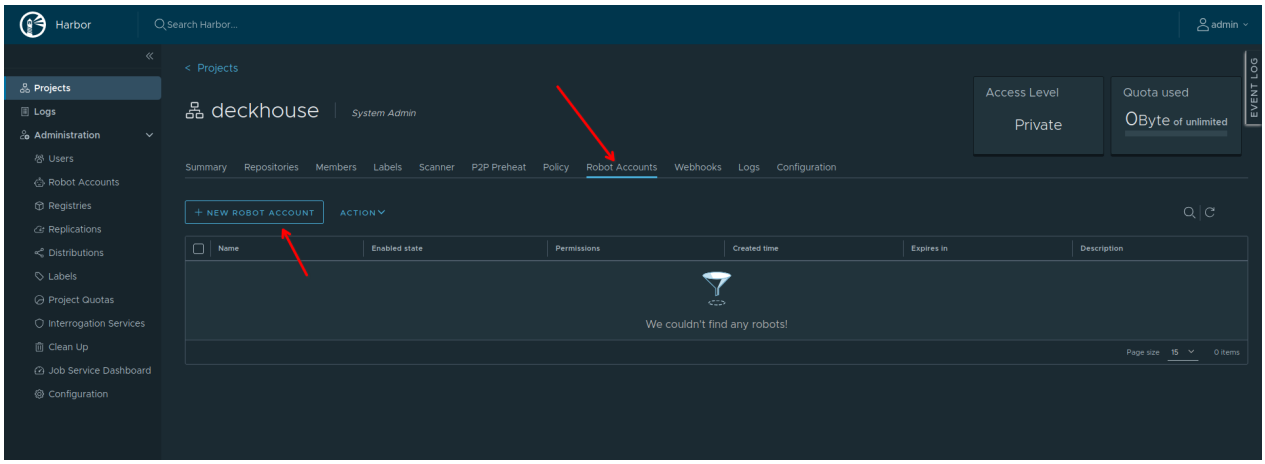
> **i** Your browser may warn about the self-signed certificate and mark the connection as “not secure”. In a private environment this is expected and acceptable. If needed, add the certificate to your browser or OS trust store to suppress the warning.

Create a new project: click **New Project**, set the name to `deckhouse`, and leave the other settings unchanged.

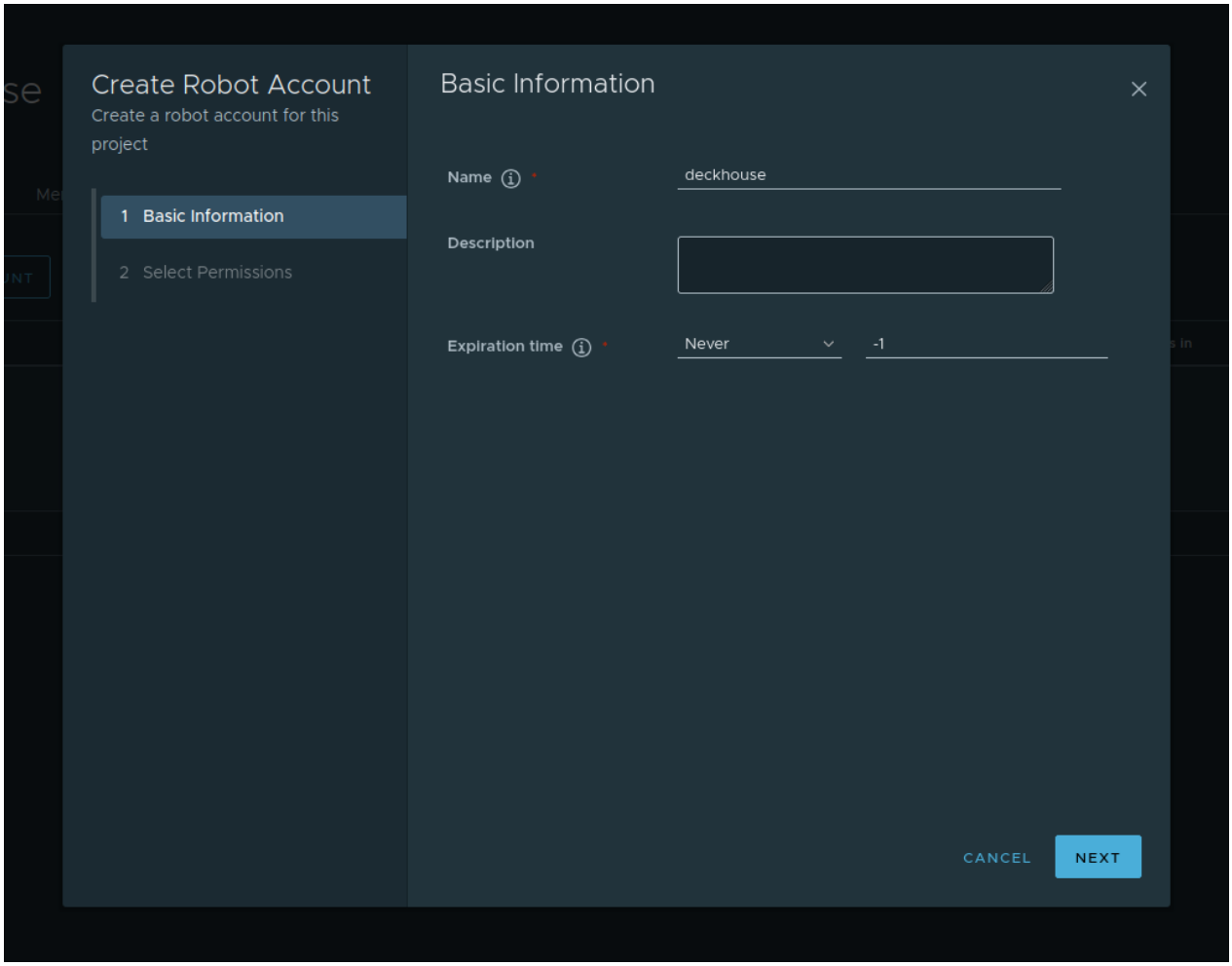


Create a robot account for this project. Robot accounts are tied to a project and meant for automation. They cannot use the web UI and are intended for Docker CLI or Helm CLI only.

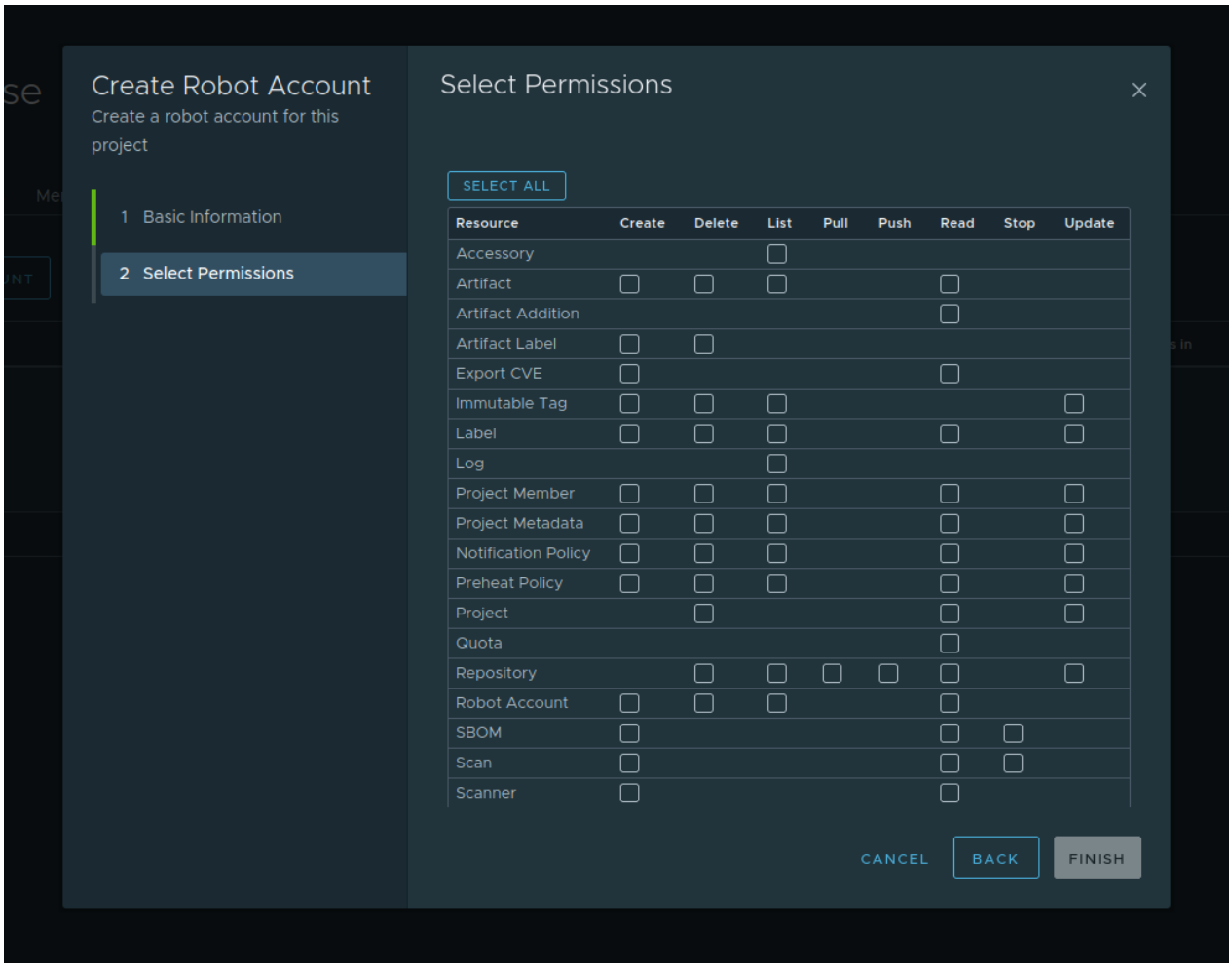
Open the `deckhouse` project and go to the **Robot Accounts** tab. Click **New Robot Account**:



Set the account name, optional description, and expiration (days or never expire):

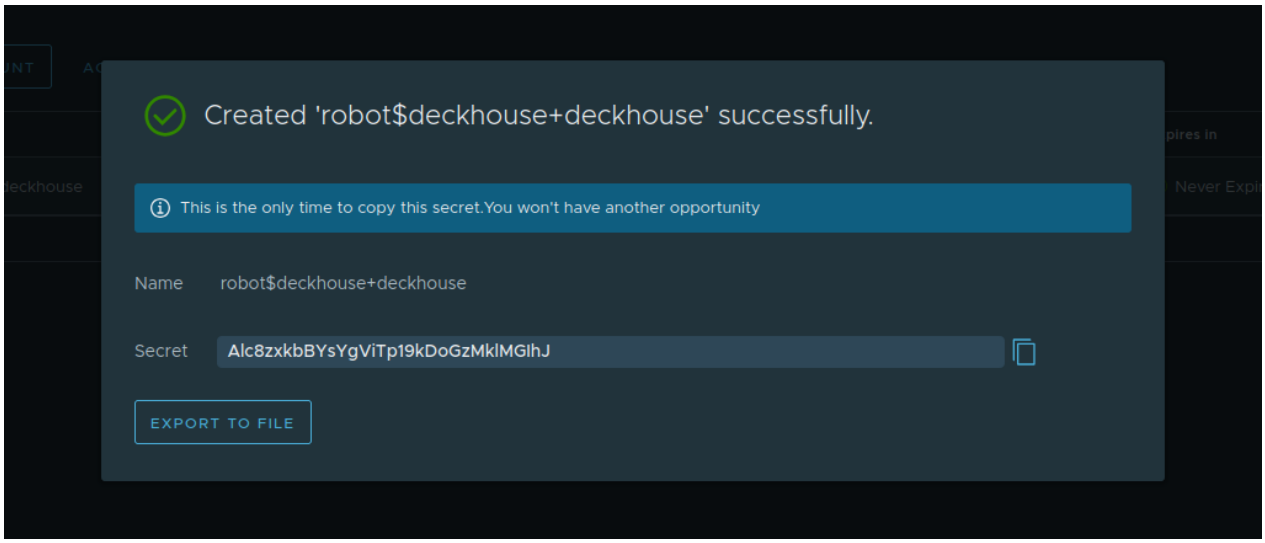


For correct operation, grant full access under **Repository**. Adjust other permissions as needed or per your security policy.



After creation, Harbor shows the robot account secret (token).

⚠ Save the secret immediately. Harbor will not show it again, and it cannot be retrieved later.



Harbor configuration is now complete.

Copying Deckhouse Platform images to a private container registry

The next step is to copy Deckhouse Platform component images from the public Deckhouse Platform registry to Harbor.

> **i** The steps in this section require the Deckhouse CLI. Install it on the host from which you will mirror images to the private registry — in this guide, the bastion host. For installation instructions, see the Deckhouse CLI documentation.

> ⚠ Downloading images takes a significant amount of time. To avoid losing progress if the SSH connection is interrupted, run the commands in a `tmux` or `screen` session. If the connection drops, you can reattach to the session and continue without starting over. Both utilities are typically available in Linux distribution repositories and can be installed using the package manager.
>
> How to use tmux...
>
> - Start a session with `tmux`.
> - Detach from the session by pressing `Ctrl + b`, then `d`. The session will keep running, and the processes started in it will continue to run. To exit the session, use `Ctrl + d`.
> - List running sessions with `tmux ls`:
>
>   ```
>   $ tmux ls
>   0: 1 windows (created Thu Dec 11 13:52:41 2025)
>   ```
>
> - Reattach to a running session: `tmux attach -t <SESSION_ID>`. In the example above, the `<SESSION_ID>` is `0`.
>
> How to use screen...
>
> - Start a session with `screen`.
> - Detach from the session by pressing `Ctrl + a`, then `d` (while holding `Ctrl`). The session will keep running, and the processes started in it will continue to run. To exit the session, use `Ctrl + d`.
> - List running sessions with `screen -r`:
>
>   ```
>   $ screen -r
>   There are several suitable screens on:
>           1166154.pts-0.guide-bastion     (12/11/25 14:00:26)     (Detached)
>           1165806.pts-0.guide-bastion     (12/11/25 13:59:35)     (Detached)
>           1165731.pts-0.guide-bastion     (12/11/25 13:59:24)     (Detached)
>           1165253.pts-0.guide-bastion     (12/11/25 13:58:16)     (Detached)
>   Type "screen [-d] -r [pid.]tty.host" to resume one of them.
>   ```
>
> - Reattach to a running session: `screen -r <SESSION_ID>`. In the example above, the `<SESSION_ID>` is `166154.pts-0.guide-bastion`.

Download Deckhouse Platform images into a dedicated directory using `d8 mirror pull`.

By default, `d8 mirror pull` downloads current Deckhouse Platform releases, vulnerability scanner databases (if included in your edition), and officially shipped modules.

Run the following command to download current image versions. Replace the placeholders with your values: `<EDITION>`, `<LICENSE_KEY>`, and optionally the target directory path:

```
d8 mirror pull --source='registry.deckhouse.io/deckhouse/<EDITION>' --
license='<LICENSE_KEY>' /home/ubuntu/d8-bundle
```

where:

- `--source` — Deckhouse Platform image registry address
- `<EDITION>` — Deckhouse Platform edition code (for example, `ee`, `se`, `se-plus`). The default is `ee` (Enterprise Edition), so `--source` may be omitted
- `--license` — Deckhouse Platform license key for authentication to the official registry
- `<LICENSE_KEY>` — your license key
- `/home/ubuntu/d8-bundle` — directory for downloaded image bundles (created automatically if missing).

If the download is interrupted, run the command again to resume, as long as no more than 24 hours have passed since it stopped.

Depending on your Internet connection, the process may take 30 to 40 minutes.

Example of a successful image download completion...

Example log when all platform components are pulled:

```
Feb 26 17:49:04.520 INFO  ╟─ [822 / 824] Pulling
registry.deckhouse.io/deckhouse/ee@sha256:4e5c17098d2a884cc971676fa9a7980f0d784a787d21e113d28a72da96ea8b2b

Feb 26 17:49:05.099 INFO  ╟─ [823 / 824] Pulling
registry.deckhouse.io/deckhouse/ee@sha256:d229564f423a1ca7a59e0be28a71218e362cc8f07d979ce63a15bb505c6ccb40

Feb 26 17:49:05.555 INFO  ╟─ [824 / 824] Pulling
registry.deckhouse.io/deckhouse/ee@sha256:eb5ed1a71783f941addf75a29e9bca9328f9dcca41d70b24b70efd4995eb1ca1

Feb 26 17:49:06.447 INFO  ╟─ All required Deckhouse images are pulled!
```

Example log when modules are packed:

```
Feb 26 18:30:18.263 INFO  ╟─ Deckhouse modules pulled!
Feb 26 18:30:18.263 INFO  ╟└ Pull images succeeded in 29m58.691782336s
Feb 26 18:30:18.265 INFO  ├ Processing image indexes
Feb 26 18:30:18.313 INFO  ├ Packing module-csi-hpe.tar
Feb 26 18:30:19.205 INFO  ├ Packing module-csi-netapp.tar
Feb 26 18:30:19.342 INFO  ├ Packing module-csi-nfs.tar
Feb 26 18:30:19.496 INFO  ├ Packing module-operator-argo.tar
Feb 26 18:30:19.666 INFO  ├ Packing module-runtime-audit-engine.tar
Feb 26 18:30:21.419 INFO  ├ Packing module-managed-memcached.tar
Feb 26 18:30:21.708 INFO  ├ Packing module-commander.tar
Feb 26 18:30:23.829 INFO  ├ Packing module-csi-s3.tar
Feb 26 18:30:25.199 INFO  ├ Packing module-csi-yadro-tatlin-unified.tar
Feb 26 18:30:25.889 INFO  ├ Packing module-operator-ceph.tar
Feb 26 18:30:27.075 INFO  ├ Packing module-operator-postgres.tar
Feb 26 18:30:28.901 INFO  ├ Packing module-pod-reloader.tar
Feb 26 18:30:28.929 INFO  ├ Packing module-sds-replicated-volume.tar
Feb 26 18:30:35.771 INFO  ├ Packing module-secrets-store-integration.tar
Feb 26 18:30:36.376 INFO  ├ Packing module-snapshot-controller.tar
Feb 26 18:30:36.458 INFO  ├ Packing module-payload-registry.tar
Feb 26 18:30:36.550 INFO  ├ Packing module-prompp.tar
Feb 26 18:30:37.318 INFO  ├ Packing module-code.tar
Feb 26 18:30:49.954 INFO  ├ Packing module-console.tar
Feb 26 18:30:50.489 INFO  ├ Packing module-csi-huawei.tar
Feb 26 18:30:53.224 INFO  ├ Packing module-storage-volume-data-manager.tar
Feb 26 18:30:53.280 INFO  ├ Packing module-operator-trivy.tar
Feb 26 18:30:54.042 INFO  ├ Packing module-sds-node-configurator.tar
Feb 26 18:30:54.249 INFO  ├ Packing module-virtualization.tar
Feb 26 18:30:58.367 INFO  ├ Packing module-commander-agent.tar
Feb 26 18:30:58.401 INFO  ├ Packing module-csi-scsi-generic.tar
Feb 26 18:31:00.045 INFO  ├ Packing module-development-platform.tar
Feb 26 18:31:00.120 INFO  ├ Packing module-sdn.tar
Feb 26 18:31:00.201 INFO  ├ Packing module-static-routing-manager.tar
Feb 26 18:31:00.228 INFO  ├ Packing module-stronghold.tar
Feb 26 18:31:01.160 INFO  ├ Packing module-sds-local-volume.tar
Feb 26 18:31:01.397 INFO  ├ Packing module-observability.tar
Feb 26 18:31:02.749 INFO  ├ Packing module-csi-ceph.tar
Feb 26 18:31:03.565 INFO  ├ Packing module-managed-postgres.tar
Feb 26 18:31:05.368 INFO  ├ Packing module-managed-valkey.tar
Feb 26 18:31:05.595 INFO  ├ Packing module-neuvector.tar
Feb 26 18:31:08.441 INFO  ├ Packing module-observability-platform.tar
Feb 26 18:31:17.443 INFO  ├ Packing module-state-snapshotter.tar
Feb 26 18:31:17.510 INFO  └ Pull Modules succeeded in 40m8.735435676s
```

Verify that the bundles were created (you should see `platform.tar`, `security.tar`, `deckhousereleases.yaml`, and multiple `module-*.tar` files):

```
$ ls -lh /home/ubuntu/d8-bundle
total 51G
-rw-rw-r-- 1 user user 4.8K Feb 26 17:19 deckhousereleases.yaml
-rw-rw-r-- 1 user user 4.9G Feb 26 18:30 module-code.tar
-rw-rw-r-- 1 user user  26G Feb 26 17:50 platform.tar
-rw-rw-r-- 1 user user 1.3G Feb 26 17:51 security.tar
```

Push the downloaded images to the private registry. Substitute the Deckhouse Platform edition and Harbor robot account credentials:

- `<ROBOT_ACCOUNT_NAME>` — robot account name
- `<PASSWORD>` — token issued when the robot account was created.

```
d8 mirror push $(pwd)/d8-bundle 'registry.local:443/deckhouse/<EDITION>' --registry-
login='robot$<ROBOT_ACCOUNT_NAME>' --registry-password='<PASSWORD>' --tls-skip-
verify
```

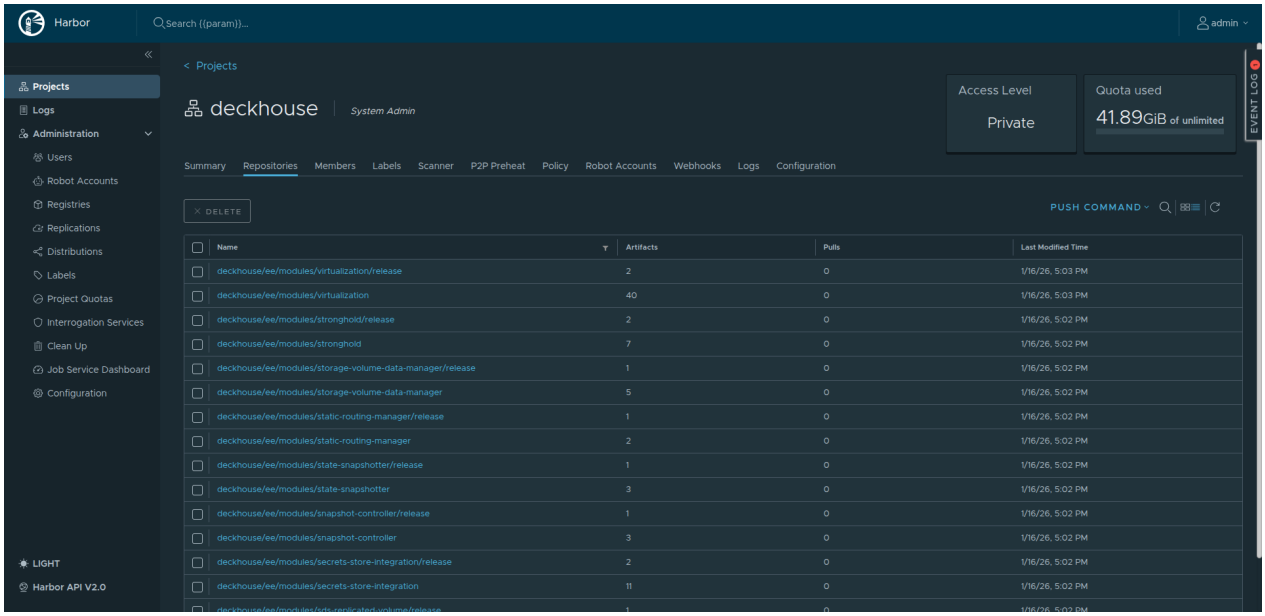
> **i** The `--tls-skip-verify` flag tells the CLI to trust the registry certificate and skip verification.

Images are read from the local bundles and pushed to the registry. This step is usually faster than download and often takes about 15 minutes.

Example of a successful image push completion...

```
Dec 11 18:25:32.350 INFO  ╟ Pushing registry.local:443/deckhouse/ee/modules/virtualization/release
Dec 11 18:25:32.351 INFO  ╟ [1 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:alpha
Dec 11 18:25:32.617 INFO  ╟ [2 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:beta
Dec 11 18:25:32.760 INFO  ╟ [3 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:early-access
Dec 11 18:25:32.895 INFO  ╟ [4 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:rock-solid
Dec 11 18:25:33.081 INFO  ╟ [5 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:stable
Dec 11 18:25:33.142 INFO  ╟ [6 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:v1.1.3
Dec 11 18:25:33.213 INFO  ╟ [7 / 7] Pushing image
registry.local:443/deckhouse/ee/modules/virtualization/release:v1.2.2
Dec 11 18:25:33.414 INFO  ╟ Pushing module tag for virtualization
Dec 11 18:25:33.837 INFO  ╚ Push module: virtualization succeeded in 43.313801312s
Dec 11 18:25:33.837 INFO   Modules pushed: code, commander-agent, commander, console, csi-ceph, csi-hpe,
csi-huawei, csi-netapp, csi-nfs, csi-s3, csi-scsi-generic, csi-yadro-tatlin-unified, development-
platform, managed-postgres, neuvector, observability-platform, observability, operator-argo, operator-
ceph, operator-postgres,
 payload-registry, pod-reloader, prompp, runtime-audit-engine, sdn, sds-local-volume, sds-node-
configurator, sds-replicated-volume, secrets-store-integration, snapshot-controller, state-snapshotter,
static-routing-manager, storage-volume-data-manager, stronghold, virtualization
```

To verify the push, open the `deckhouse` project in the Harbor web UI.



The images are now available in the private container registry.

Signing in to the registry to run the installer

Use the host from which you will run the installer (in this guide, the bastion host). On that host, `registry.local` must resolve to the Harbor VM (via `/etc/hosts` or DNS). Configure Docker to trust the TLS registry the same way as on the Harbor host: create `/etc/docker/certs.d/registry.local/` and place the required certificates there (copy them from the Harbor VM or generate again).

Sign in to Harbor so Docker can pull the dhctl installer image:

```
docker login registry.local
```

Example of a successful command execution...

```
$ docker login registry.local
Username: deckhouse
Password:

WARNING! Your credentials are stored unencrypted in '/home/ubuntu/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
```

Preparing VMs for the future nodes

VM requirements

⚠ The example configuration in this guide sets the `ContainerdV2` container runtime (`defaultCRI: ContainerdV2`). To use it, the nodes must meet the following requirements:

- `CgroupsV2` support;
- systemd version `244` or newer;
- Linux kernel version `5.8` or newer, except for versions 6.12.0–6.12.28 and 6.14.0–6.14.6;
- support for the `erofs` kernel module.

Some distributions (for example, Astra Linux 1.7.4) do not meet these requirements. Bring the node OS into compliance before installing Deckhouse Platform. For details, see the defaultCRI parameter description.

Servers intended for future cluster nodes must meet the following requirements:

- at least 4 CPU cores;
- at least 8 GB RAM;
- at least 60 GB of disk space on fast storage (400+ IOPS);
- a supported OS;
- Linux kernel version `5.8` or later;
- a **unique hostname** across all cluster servers (physical servers and virtual machines);
- one of the package managers available (`apt`/`apt-get`, `yum`, or `rpm`).
- Python installed;
- access to the proxying registry or to the private container registry that holds Deckhouse Platform images;
- access to the standard OS package repositories for your distribution (via a proxy or an internal package mirror);
- SSH access from the bastion host using a key;
- network access from the bastion host on port `22/TCP`;
- no container runtime packages installed on the node (for example, no containerd or Docker).

⚠ For proper resource sizing, read the production preparation guide and the system requirements for deployment options, node types, and sizing based on workload and operations.

Mapping registry.local to the Harbor VM

On the servers where the master and worker nodes will run, make the `registry.local` hostname resolve to the Harbor VM’s internal IP address in the private network.

Connect to each server in turn and add a line to `/etc/hosts` (and, if your cloud provider manages the file, update the cloud template as well).

How to connect to a server without external access...

To connect via SSH to a server without external access, you can use the Bastion host as a jump host.

There are two ways to connect:

1. *Connect via a jump host.* Run the command:

   ```
   ssh -J ubuntu@<BASTION_IP> ubuntu@<NODE_IP>
   ```

   In this mode, you first connect to the Bastion host, and then connect through it to the target server using the same SSH key.

2. *Connect with agent forwarding.* Connect to the Bastion host using:

   ```
   ssh -A ubuntu@<BASTION_IP>
   ```

   > **i** Note: for this to work, you may need to start ssh-agent and add your key with `ssh-add` on the workstation from which you run the command.

   Then connect to the target servers:

   ```
   ssh ubuntu@<NODE_IP>
   ```

```
<INTERNAL_IP_ADDRESS> registry.local proxy.local
```

> **i** Replace `<INTERNAL_IP_ADDRESS>` with the Harbor VM's actual internal IP address.

Configuring SSH access to the master node

To install Deckhouse Platform, add the public part of your SSH key to `.ssh/authorized_keys` for the user on the future master node that will be used to install the platform.

Run the commands as `root` (substitute the public part of your SSH key):

```
mkdir /home/<USER>/.ssh
export KEY='ssh-rsa AAAAB3NzaC1yc2EAAAADA...'
echo $KEY >> /home/<USER>/.ssh/authorized_keys
chmod 700 /home/<USER>/.ssh
chmod 600 /home/<USER>/.ssh/authorized_keys
echo '<USER> ALL=(ALL) NOPASSWD: ALL' | sudo EDITOR='tee -a' visudo
```

Here `<USER>` is the username on the master node.

How to obtain the public part of the key...

Run `cat ~/.ssh/id_rsa.pub` to print the public key (or use the path to your key's `.pub` file).

These commands add the public SSH key so you can log in to the server as `<USER>`, and configure passwordless privilege escalation for this user.

Verify that you can connect as the user:

```
ssh -J ubuntu@<BASTION_IP> <USER>@<NODE_IP>
```

Creating a user for the worker node

> **i** The following prepares the node for Cluster API Provider Static (CAPS). If you prefer to add static nodes manually with the bootstrap script, you can skip this subsection and the later CAPS steps: create a `Static` NodeGroup, take the script from the Secret, and run it on the server as described in the documentation (manual method).

On the **master node**, generate an SSH key with an empty passphrase:

```
ssh-keygen -t rsa -f /dev/shm/caps-id -C "" -N ""
```

On the worker node server, create the `caps` user. Run the following commands and set the public key from the previous step:

```
# Set the user’s public SSH key.
export KEY='<SSH_PUBLIC_KEY>'
useradd -m -s /bin/bash caps
usermod -aG sudo caps
echo 'caps ALL=(ALL) NOPASSWD: ALL' | sudo EDITOR='tee -a' visudo
mkdir /home/caps/.ssh
echo $KEY >> /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
```

If you are using CentOS, Rocky Linux, ALT Linux, ROSA Server, RED OS, or MOS OS...

On RHEL-based operating systems, add the `caps` user to the `wheel` group. Run the following commands and set the public key from the previous step:

```
# Set the user’s public SSH key.
export KEY='<SSH_PUBLIC_KEY>'
useradd -m -s /bin/bash caps
usermod -aG wheel caps
echo 'caps ALL=(ALL) NOPASSWD: ALL' | sudo EDITOR='tee -a' visudo
mkdir /home/caps/.ssh
echo $KEY >> /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
```

If you are using an Astra Linux family OS...

On Astra Linux systems that use the Parsec mandatory integrity control module, set the maximum integrity level for the `caps` user:

```
pdpl-user -i 63 caps
```

Preparing the configuration file

The configuration file for a private environment differs from bare metal in several ways. Take `config.yml` from step 4 of the bare metal guide and apply the changes below.

If cluster nodes need outbound access via a proxy, deploy the proxy ahead of time—preferably on a dedicated machine with Internet access.

Example: Squid proxy in a container...

A proxy may be required for traffic such as pulling images from the public Deckhouse Platform registry to the bastion, or for Deckhouse Platform components and nodes to reach external URLs when policy allows. OS packages on nodes can still come from internal mirrors, in which case the proxy is not used for package traffic.

Deploy a proxy on a separate machine if your environment allows external access.

You may use any suitable proxy. This example uses Squid:

```
docker run -d --name squid -p 3128:3128 ubuntu/squid
```

Example of a successful start:

```
$ docker run -d --name squid -p 3128:3128 ubuntu/squid
Unable to find image 'ubuntu/squid:latest' locally
latest: Pulling from ubuntu/squid
1678e6c91c57: Pull complete
040467b888ae: Pull complete
18b9e99f4452: Pull complete
Digest: sha256:6a097f68bae708cedbabd6188d68c7e2e7a38cedd05a176e1cc0ba29e3bbe029
Status: Downloaded newer image for ubuntu/squid:latest
059b21fddbd2aba33500920f3f6f0712fa7b23893d512a807397af5eec27fb37
```

Check that the container is running:

```
059b21fddbd2   ubuntu/squid                           "entrypoint.sh -f /e…"   About
a minute ago   Up About a minute     0.0.0.0:3128->3128/tcp, [::]:3128->3128/tcp
squid
```

You should see a container named `squid` in the list.

- In ClusterConfiguration, set proxy parameters **if** the environment uses a proxy for external access:

  ```
  # Proxy server settings.
  proxy:
    httpProxy: http://proxy.local:3128
    httpsProxy: https://proxy.local:3128
    noProxy: ["registry.local", "proxy.local", "10.128.0.8", "10.128.0.32",
  "10.128.0.18"]
  ```

  Here you specify:

  - HTTP and HTTPS proxy addresses
  - hostnames and IP addresses that **must not** use the proxy (internal names and internal IPs of your servers).
- In the `deckhouse` ModuleConfig:
  - Set `releaseChannel` to `Stable` to use the stable release channel.
  - In the `spec.settings.registry` section, specify access settings for the private container registry (Harbor in this guide):

    ```
    # Settings for accessing the container registry with Deckhouse Platform
    images.
    registry:
      mode: Unmanaged
      unmanaged:
        # Address of the registry.
        imagesRepo: <IMAGES_REPO_URI>
        # Username for authenticating with the registry.
        username: <REGISTRY_USERNAME>
        # Password for authenticating with the registry.
        password: <REGISTRY_PASSWORD>
        scheme: HTTPS
        # Root CA certificate in PEM format used to verify the registry server
    certificate.
        ca: <REGISTRY_CA>
    ```
- In the global ModuleConfig, enable self-signed certificates for modules and set `publicDomainTemplate` for system application hostnames:

```
settings:
  modules:
    # Template for system application URLs in the cluster.
    # With %s.test.local, Grafana is served at grafana.test.local.
    # MUST NOT match network.clusterDomain of the control-plane-manager module.
    # You may change this now or after installation.
    publicDomainTemplate: "%s.test.local"
    # How Deckhouse Platform modules terminate HTTPS.
    https:
      certManager:
        clusterIssuerName: selfsigned
```

The `settings.modules.https` block in ModuleConfig/global supports several modes: `CertManager` (certificate from the chosen `ClusterIssuer` — not necessarily `selfsigned`; can be corporate CA, HashiCorp Vault, Venafi, etc., see the certificate overview); `CustomCertificate` (TLS Secret in `d8-system`); with an external TLS terminator, `OnlyInURI` is possible. Using `selfsigned` together with disabling Let's Encrypt below is a simple pattern for isolated environments without ACME.

- In the `user-authn` ModuleConfig, set dexCAMode to `FromIngressSecret`:

```
settings:
  controlPlaneConfigurator:
    dexCAMode: FromIngressSecret
```

- Enable `cert-manager` and disable Let's Encrypt:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  version: 1
  enabled: true
  settings:
    disableLetsencrypt: true
```

- In StaticClusterConfiguration, set `internalNetworkCIDRs` to the subnet of the nodes' internal IPs. For example:

```
internalNetworkCIDRs:
  - 10.128.0.0/24
```

Full configuration file example...

```
# Cluster-wide settings.
# https://deckhouse.io/products/kubernetes-
platform/documentation/v1/reference/api/cr.html#clusterconfiguration
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Static
# The default container runtime type used on cluster nodes (in NodeGroups).
defaultCRI: "ContainerdV2"
# Proxy server settings.
proxy:
  httpProxy: http://proxy.local:3128
  httpsProxy: https://proxy.local:3128
  noProxy: ["registry.local", "proxy.local", "10.128.0.8", "10.128.0.32",
"10.128.0.18"]
---
# control-plane-manager module settings.
# https://deckhouse.io/modules/control-plane-manager/configuration.html
```

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "Default"
    network:
      # Cluster Pod address space.
      # You may want to change this. Make sure it does not overlap with
serviceSubnetCIDR and internalNetworkCIDRs.
      podSubnetCIDR: 10.111.0.0/16
      # Cluster Service address space.
      # You may want to change this. Make sure it does not overlap with podSubnetCIDR
and internalNetworkCIDRs.
      serviceSubnetCIDR: 10.222.0.0/16
      # Cluster domain.
      clusterDomain: "cluster.local"
    apiserver:
      # Enable access to the Kubernetes API server via Ingress.
      # https://deckhouse.io/modules/control-plane-
manager/configuration.html#parameters-apiserver-publishapi
      publishAPI:
        ingress:
          enabled: true
          https:
            mode: Global
            global:
              kubeconfigGeneratorMasterCA: ""
---
# deckhouse module settings.
# https://deckhouse.io/modules/deckhouse/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    bundle: Default
    releaseChannel: Stable
    logLevel: Info
    # Settings for accessing the container registry with Deckhouse Platform images.
    registry:
      mode: Unmanaged
      unmanaged:
        # Address of the Docker registry where the Deckhouse Platform images are
located.
        imagesRepo: <IMAGES_REPO_URI>
        # The username for authenticating with the container registry.
        username: <USERNAME>
        # The password for authenticating with the container image repository.
        password: <PASSWORD>
        scheme: HTTPS
        # The root CA certificate (in PEM format) for validating the container
registry’s server certificate.
        ca: |
          -----BEGIN CERTIFICATE-----
          ...
```

```
        -----END CERTIFICATE-----
---
# Global Deckhouse Platform settings.
# https://deckhouse.io/products/kubernetes-
platform/documentation/v1/reference/api/global.html#%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D
0%B5%D1%82%D1%80%D1%8B
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings:
    modules:
      # A template used to construct the addresses of system applications in the
cluster.
      # For example, with %s.test.local, Grafana will be available at
'grafana.test.local'.
      # The domain MUST NOT match the value specified in the network.clusterDomain
parameter of the control-plane-manager module.
      # You can set your own value right away, or follow the guide and change it
after installation.
      publicDomainTemplate: "%s.test.local"
      # The HTTPS implementation method used by Deckhouse Platform modules.
      https:
        certManager:
          # Use self-signed certificates for Deckhouse Platform modules.
          clusterIssuerName: selfsigned
---
# user-authn module settings.
# https://deckhouse.io/modules/user-authn/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: FromIngressSecret
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  version: 1
  enabled: true
  settings:
    disableLetsencrypt: true
---
# cni-cilium module settings.
# https://deckhouse.io/modules/cni-cilium/configuration.html
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  # Enable the cni-cilium module.
  enabled: true
```

```
  settings:
    # cni-cilium module settings.
    # https://deckhouse.io/modules/cni-cilium/configuration.html
    tunnelMode: VXLAN
---
# Static cluster settings.
# https://deckhouse.io/products/kubernetes-
platform/documentation/v1/reference/api/cr.html#staticclusterconfiguration
apiVersion: deckhouse.io/v1
kind: StaticClusterConfiguration
# A list of internal node networks (for example, '10.0.4.0/24') used for
communication between Kubernetes components
# (kube-apiserver, kubelet, etc.).
# Specify this if you use the virtualization module or if cluster nodes have more
than one network interface.
# If cluster nodes use only one interface, you can omit the
StaticClusterConfiguration resource.
internalNetworkCIDRs:
  - 10.128.0.0/24
```

The installation configuration file is ready.

Installing Deckhouse Platform

Copy the prepared configuration file to the host from which you run the installation (for example, `~/deckhouse` on the bastion). Go to that directory and start the installer:

```
docker run --pull=always -it -v "$PWD/config.yml:/config.yml" -v
"$HOME/.ssh/:/tmp/.ssh/" --network=host -v "$PWD/dhctl-tmp:/tmp/dhctl"
registry.local/deckhouse/<EDITION>/install:stable bash
```

If you get the `509: certificate signed by unknown authority` error...

Even if the certificates are present in `/etc/docker/certs.d/registry.local/`, Docker may still report that the certificate is signed by an unknown certificate authority (which is typical for self-signed certificates). In most cases, adding `ca.crt` to the system trusted certificate store and restarting Docker resolves the issue.

> **i** If there is no internal DNS server and hostnames are only in `/etc/hosts` on the machine where you start the installer, use `--network=host` so Docker can resolve those names.

After the image is pulled and the container starts successfully, you will see a shell prompt inside the container:

```
[deckhouse] root@guide-bastion / #
```

Start the Deckhouse Platform installation with the following command (specify the master node's internal IP address):

```
dhctl bootstrap --ssh-user=<USER> --ssh-host=<MASTER_IP> --ssh-agent-private-
keys=/tmp/.ssh/id_rsa --config=/config.yml
```

Here:

- `<USER>`: User on the master node for whom you configured SSH access and passwordless `sudo`.
- `<MASTER_IP>`: Internal IP address of the master node.

Replace `id_rsa` with the name of your private key file if it differs.

The installation process may take up to 30 minutes depending on the network speed.

If the installation completes successfully, you will see the following message:

```
┌ ~ Bootstrap: Run post bootstrap actions
│ ┌ Set release channel to deckhouse module config
│ │  Succeeded!
│ └ Set release channel to deckhouse module config (0.09 seconds)
└ ~ Bootstrap: Run post bootstrap actions (0.09 seconds)

┌ ~ Bootstrap: Clear cache
│ ~ Next run of "dhctl bootstrap" will create a new Kubernetes cluster.
└ ~ Bootstrap: Clear cache (0.00 seconds)

 Deckhouse cluster was created successfully!
```

Adding nodes to the cluster

Add a worker node to the cluster.

Perform the following steps:

- Configure a StorageClass for local storage by running the following command on the master node:

  ```
  sudo -i d8 k create -f - << EOF
  apiVersion: deckhouse.io/v1alpha1
  kind: LocalPathProvisioner
  metadata:
    name: localpath
  spec:
    path: "/opt/local-path-provisioner"
    reclaimPolicy: Delete
  EOF
  ```

- Set the created StorageClass as the default StorageClass. To do this, run the following command on the master node:

  ```
  sudo -i d8 k patch mc global --type merge -p "{\"spec\": {\"settings\":
  {\"defaultClusterStorageClass\":\"localpath\"}}}"
  ```

- Create the `worker` NodeGroup and add a node using Cluster API Provider Static (CAPS):

  ```
  sudo -i d8 k create -f - <<EOF
  apiVersion: deckhouse.io/v1
  kind: NodeGroup
  metadata:
    name: worker
  spec:
    nodeType: Static
    staticInstances:
      count: 1
      labelSelector:
        matchLabels:
          role: worker
  EOF
  ```

- Create an SSHCredentials resource in the cluster. Run on the master node:

```
sudo -i d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha2
kind: SSHCredentials
metadata:
  name: caps
spec:
  user: caps
  privateSSHKey: "`cat /dev/shm/caps-id | base64 -w0`"
EOF
```

- Print the public SSH key (needed for verification). On the master node:

```
cat /dev/shm/caps-id.pub
```

- Create a StaticInstance for the node to add. On the master node, set the node IP and apply:

```
# Specify the IP address of the node to be added to the cluster.
export NODE=<NODE_IP_ADDRESS>
sudo -i d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha2
kind: StaticInstance
metadata:
  name: d8cluster-worker
  labels:
    role: worker
spec:
  address: "$NODE"
  credentialsRef:
    kind: SSHCredentials
    name: caps
EOF
```

- Make sure all cluster nodes are in the `Ready` status:

```
$ sudo -i d8 k get no
NAME               STATUS   ROLES                  AGE    VERSION
d8cluster          Ready    control-plane,master   30m   v1.34.10
d8cluster-worker   Ready    worker                 10m   v1.34.10
```

  It may take some time for all Deckhouse Platform components to start after the installation completes.

Configuring the Ingress controller and creating a user

Installing the ingress controller

Make sure the Kruise controller manager Pod of the ingress-nginx module is running and in the `Running` status. To do this, run the following command on the master node:

```
$ sudo -i d8 k -n d8-ingress-nginx get po -l app=kruise
NAME                                          READY   STATUS    RESTARTS    AGE
kruise-controller-manager-7dfcbdc549-b4wk7   3/3     Running   0           15m
```

Create the `ingress-nginx-controller.yml` file on the master node containing the Ingress controller configuration:

```
# Ingress NGINX controller parameters.
# https://deckhouse.io/modules/ingress-nginx/cr.html
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  # The name of the IngressClass served by the Ingress NGINX controller.
  ingressClass: nginx
  # How traffic enters from outside the cluster.
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  # Defines which nodes will run the component.
  # You may want to adjust this.
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/control-plane
      operator: Exists
```

Apply it by running the following command on the master node:

```
sudo -i d8 k create -f $PWD/ingress-nginx-controller.yml
```

Starting the Ingress controller after Deckhouse Platform installation may take some time. Before you proceed, make sure the Ingress controller is running (run the following command on the master node):

```
$ sudo -i d8 k -n d8-ingress-nginx get po -l app=controller
NAME                                          READY   STATUS    RESTARTS   AGE
controller-nginx-r6hxc                        3/3     Running   0          5m
```

Creating a user to access the cluster web-interface

Create the `user.yml` file on the master node containing the user account, group, and access rights:

```
# Static user data.
# https://deckhouse.io/modules/user-authn/cr.html#user
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  # User email.
  email: admin@deckhouse.io
  # Generate password hash using the following command and replace <PASSWORD_HASH>
with the output:
  # echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64
-w0; echo
  password: '<PASSWORD_HASH>'
---
# User group.
# https://deckhouse.io/modules/user-authn/cr.html#group
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
  - kind: User
    name: admin
---
# RBAC and authorization settings.
# https://deckhouse.io/modules/user-authz/cr.html#clusterauthorizationrule
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  # List of Kubernetes RBAC subjects.
  subjects:
  - kind: Group
    name: admins
  # A predefined access level template.
  accessLevel: SuperAdmin
  # Allow the user to use kubectl port-forward.
  portForwarding: true
```

Apply it by running the following command on the master node:

```
sudo -i d8 k create -f $PWD/user.yml
```

Configuring DNS records

To reach the cluster web UIs, make the hostnames below resolve to the master node’s internal IP address. Names must follow the `publicDomainTemplate` you configured (here, `%s.test.local`). Replace `<MASTER_IP>` with the master’s internal IP before running:

```
export PUBLIC_IP="<MASTER_IP>"
sudo -E bash -c "cat <<EOF >> /etc/hosts
$PUBLIC_IP api.test.local
$PUBLIC_IP code.test.local
$PUBLIC_IP commander.test.local
$PUBLIC_IP registry.test.local
$PUBLIC_IP console.test.local
$PUBLIC_IP dex.test.local
$PUBLIC_IP documentation.test.local
$PUBLIC_IP grafana.test.local
$PUBLIC_IP hubble.test.local
$PUBLIC_IP istio.test.local
$PUBLIC_IP istio-api-proxy.test.local
$PUBLIC_IP kubeconfig.test.local
$PUBLIC_IP openvpn-admin.test.local
$PUBLIC_IP prometheus.test.local
$PUBLIC_IP status.test.local
$PUBLIC_IP upmeter.test.local
EOF
"
```

To confirm the cluster is healthy, open Grafana (built from `publicDomainTemplate`, e.g. `grafana.test.local` for `%s.test.local`) and sign in with the user you created earlier.

Next steps

Deckhouse Platform is installed and running. You can use the web UIs to manage the cluster:

- **Deckhouse Console** — cluster and core component management. URL: **console.test.local**. You can download the `d8` CLI from this UI.
- **Documentation** — documentation for the Deckhouse Platform version running in the cluster. URL: **documentation.test.local**.
- **Monitoring** — Grafana dashboards shipped with Deckhouse Platform. URL: **grafana.test.local** (Prometheus UI path: **/prometheus/**). More in the monitoring documentation.
- **Status page** — overall Deckhouse Platform and component status. URL: **status.test.local**.
- **Upmeter** — SLA tracking by component and period. URL: **upmeter.test.local**.
- **Production readiness** — follow the production preparation guide before taking real traffic.

Deploying your first application

- **CI/CD access** — create a ServiceAccount for deployments and grant RBAC to obtain a kubeconfig for automation. See CI/CD access section. URL: **kubeconfig.test.local**.
- **Routing traffic to an app** — create a Service and Ingress. See Ingress and incoming traffic section.
- **Application monitoring** — add annotations `prometheus.deckhouse.io/custom-target: "my-app"` and `prometheus.deckhouse.io/port: "80"` to the Service. See Application and infrastructure monitoring section.

Learn more

More about Deckhouse Platform is in the documentation. For questions, join the community on Telegram.

Additional resources

- Global custom resources
- Global parameters
- Module `cert-manager` documentation
- Module `deckhouse` parameters
- Module `registry` documentation
- Module `user-authn` parameters

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Switching CNI in the cluster

This document describes the procedure for changing the network plugin (CNI) in a Deckhouse Platform (DP) cluster. The tool used in DP allows performing the automated migration (e.g., from Flannel to Cilium) with minimal application downtime and without a full restart of the cluster nodes.

- This guide is applicable for DP version 1.76 and above. For DP version 1.75 and earlier, use the Switching CNI from Flannel or Simple bridge to Cilium guide.
- The tool is not intended for switching to any (third-party) CNI.
- During the migration process, the target CNI module (`ModuleConfig.spec.enabled: true`) will be automatically enabled, which must be pre-configured by the cluster administrator.

- During the migration process, all pods in the cluster using the network (in PodNetwork) created by the current CNI will be restarted. This will cause an interruption in service availability. To minimize the risks of losing critical data, it is highly recommended to stop the operation of the most critical application services yourself before carrying out the work.
- It is recommended to carry out work during an agreed maintenance window.
- Before carrying out work, it is necessary to disable external cluster management systems (CI/CD, GitOps, ArgoCD, etc.) that may conflict with the process (e.g., trying to restore deleted pods prematurely or rolling back settings). Also, ensure that the cluster management system does not enable the old CNI module.

Supported CNI switching modes (rows are the current CNI, columns are the target CNI):

Current CNI	simple-bridge	flannel (hostgw)	flannel (vxlan)	cilium (native)	cilium (vxlan)
simple-bridge	□	□	□	□	□
flannel (hostgw)	□	□	□	□	□
flannel (vxlan)	□	□	□	□	□
cilium (native)	□	□	□	□	□
cilium (vxlan)	□	□	□	□	□

There are several methods to switch CNI in a DP cluster.

Method 1: Using the d8 network cni-migration command group of the d8 utility (automated switching)

The d8 utility provides a command group for managing the migration process.

Starting the migration

To start the process, execute the `switch` command, specifying the target CNI (e.g., `cilium`, `flannel`, or `simple-bridge`):

```
d8 network cni-migration switch --to-cni cilium
```

This command will create the necessary resource in the cluster and start the migration controller. DP will automatically deploy the necessary components: Manager and Agents in the `d8-system` namespace.

Monitoring progress

To monitor the progress in real-time, use the command:

```
d8 network cni-migration watch
```

You will see a dynamic interface with the following information:

- **Current phase**: What exactly is happening at the moment (e.g., `CleaningNodes` or `RestartingPods`).
- **Progress**: List of successfully completed stages and current status of pending cluster actions.
- **Errors**: If a problem occurs on any node, it will be displayed in the `Failed Nodes` list.

Main phases of the process:

1. **Preparing**: Validating the request and waiting for the environment to be ready (e.g., webhooks disabled).
2. **WaitingForAgents**: Waiting for migration agents to start on all nodes.
3. **EnablingTargetCNI**: Enabling the target CNI module in the DP configuration.
4. **DisablingCurrentCNI**: Disabling the current CNI module.
5. **CleaningNodes**: Agents clean up the network settings of the current CNI on the nodes.
6. **WaitingTargetCNI**: Waiting for the new CNI pods (DaemonSet) to be ready.
7. **RestartingPods**: Restarting application pods to switch them to the new network.
8. **Completed**: Migration successfully completed.

Completion and cleanup

After the migration status changes to `Succeeded`, you must remove the migration resources (controllers and agents) so that they do not consume cluster resources. To do this, use the command:

```
d8 network cni-migration cleanup
```

Method 2: Using the d8 k commands (manual switching)

The user has the option to manage the migration directly via the Kubernetes API.

Starting the migration

Create a CNIMigration resource (the example uses the `cni-migration.yaml` manifest) specifying the target CNI:

```
---
apiVersion: network.deckhouse.io/v1alpha1
kind: CNIMigration
metadata:
  name: migration-to-cilium
spec:
  targetCNI: cilium
```

Apply the manifest in the cluster:

```
d8 k create -f cni-migration.yaml
```

Monitoring progress

Monitor the status of the CNIMigration resource:

```
d8 k get cnimigration migration-to-cilium -o yaml -w
# OR
watch -n 1 "d8 k get cnimigration migration-to-cilium -o yaml"
```

Pay attention to the fields:

- `status.phase`: Current stage.
- `status.conditions`: Detailed history of transitions.
- `status.failedSummary`: List of nodes with errors.

For detailed diagnostics of a specific node, you can check its local resource:

```
d8 k get cninodemigrations
d8 k get cninodemigration NODE_NAME -o yaml
```

To view the logs of migration controllers in the cluster, execute the following commands:

```
d8 k -n d8-system get pods -o wide | grep cni-migration
d8 k -n d8-system logs cni-migration-manager-HASH
d8 k -n d8-system logs cni-migration-agent-HASH
```

Completion and cleanup

After successful completion (the `CNIMigration` status shows condition `Type: Succeeded, Status: True`), delete the resource:

```
d8 k delete cnimigration migration-to-cilium
```

This action signals DP to remove all previously created resources in the cluster.

Troubleshooting

> **i** The CNI switching tool does not evaluate the network connectivity of pods and cluster components after the CNI migration in the cluster.

Agent does not start on a node

Check the status of the `cni-migration-agent` DaemonSet in the `d8-system` namespace. There may be taints on the node that are not covered by the agent's tolerations.

Node stuck in CleaningNodes phase

Check the logs of the agent pod on the corresponding node:

```
d8 k -n d8-system logs cni-migration-agent-HASH
```

Possible reason: inability to delete CNI configuration files due to permissions, stuck processes, or failure to pass Webhooks verification.

Target CNI pods do not start

If the target CNI (e.g., Cilium) is in `Init:0/1` status, check the logs of its init container `cni-migration-init-checker`. It waits for the node cleanup to complete. If cleanup is not finished (see the previous section), the new CNI will not start. In a critical situation, you can edit the Daemonset to remove the `cni-migration-init-checker` init container.

Migration stuck

If the process has stopped and hasn't moved for a long time:

1. Check `failedSummary` in the `CNIMigration` status.
2. If there are problematic nodes that cannot be fixed (e.g., a node in NotReady status), you can temporarily remove this node from the cluster or try to reboot it.

Additional resources

- Global custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Planning for a high pod count on a node

When you plan to place and run more than 100 pods on a single node, a number of additional requirements apply to its configuration. The most critical requirements apply to the disk (unpacking and mounting image layers), kernel, and RAM. Following the recommendations in this guide helps reduce pod startup time and make it more predictable.

> **i** For nodes with high pod density:
>
> - Plan the node subnet size in advance (via `podSubnetNodeCIDRPrefix`) — the pod limit per node will adjust automatically.
> - Use a Linux kernel version 6.12 or newer.
> - Use a fast local disk (NVMe or SSD) and do not use a network file system.
> - Use lightweight application images with minimum amount of layers (for example, `distroless`).

For such scenarios, Deckhouse Platform (DP) configures control plane components automatically — no additional manual configuration is required.

Pod limit per node

> **i** The pod limit per node in DP is calculated based on the node subnet size set via the `podSubnetNodeCIDRPrefix` parameter. To fit up to 1000 pods on a node, set the value ≤ `21` when deploying the cluster.

The subnet size allocated to a node determines the maximum number of pods that can be allocated to a node:

Value of `podSubnetNodeCIDRPrefix`	Pods per node (by default)
≥ `24`	120
`23`	250
`22`	500
≤ `21`	1000

The `podSubnetNodeCIDRPrefix` parameter is set when the cluster is deployed. If you know that you will be allocating a large number of pods on nodes, it is recommended that you set an appropriate value when the cluster is created.

Node kernel

> **i** For nodes with a high pod count, it is recommended that you use a Linux kernel version 6.12 or newer with support for the `erofs` kernel module.

Starting with version 2.x, containerd supports `EROFS` functionality that depends on the `CONFIG_EROFS_FS_BACKED_BY_FILE` kernel option. This option is enabled by default starting with Linux 6.12 and allows mounting image layers represented as regular files directly, without creating a separate loop device for each layer or container.

Without this option, starting a large number of containers can create many loop devices. On nodes with hundreds or thousands of pods, this increases system load and can significantly slow down container startup.

Therefore, for nodes where you plan to run a large number of pods, it is recommended that you use an operating system with a Linux kernel 6.12 or newer.

Node disk

> **i** Use a fast local disk (NVMe or SSD). Do not place container runtime files and kubelet data on a network file system.

Starting each pod involves disk-intensive operations of unpacking and working with image layers. During a mass startup, these operations are multiplied by the number of pods, so disk performance directly affects startup time.

Recommendations:

- Use fast disks with performance of at least 400 IOPS (see the “Resource requirements” section). For nodes with high pod density, prioritize local NVMe disks.
- Do not place container runtime and kubelet directories in a network or distributed file system. Network latency on mount operations is multiplied by the number of started pods and makes startup time unpredictable.

Application images

> **i** The more compact and simple the application image, the faster the pod starts. Use minimal images (for example, `distroless`) and reduce the number of layers.

Image size and structure affect its pull and preparation time. With a large number of simultaneously started pods, this can become a problem.

Recommendations:

- Use minimal base images (for example, `distroless`, `*-slim`, or `alpine`), without unnecessary packages and tools.
- Reduce the number of image layers and merge them when possible to decrease the amount of unpacking and mounting.
- Keep images small — remove package manager caches, build artifacts, and temporary files.
- Use multi-stage builds to reduce the size and number of layers in the final image.

Pod startup delays can also be caused not by the image itself, but by the container registry and the network bandwidth to it. During a mass startup or update of multiple applications, a node may download several images at once, which can make the registry a bottleneck.

Recommendations:

- Place the container registry close to the cluster in network terms and ensure sufficient bandwidth to it.
- For images from external (including public) container registries, use a local mirror or pull-through cache near the cluster.

Node resources

In addition to disk and kernel, pod density is limited by the node’s CPU and RAM. The following are test results where a static NGINX is used:

- A node with 4 vCPU, 8 GB RAM, and an SSD ensures a simultaneous startup of 150 pods.
- A node with 32 vCPU, 64 GB RAM, local NVMe, and a Linux kernel ≥ 6.12 starts about 1000 pods in a few minutes.

Actual values depend on application characteristics, so it is recommended that you conduct a load testing before using similar configurations in a production environment.

Additional resources

- Module `control-plane-manager` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Installing Deckhouse Platform in an existing Talos cluster

This guide applies to an existing, operational Kubernetes cluster based on Talos Linux: the control plane is running, worker nodes have joined the cluster, a CNI is installed, and the Kubernetes API is accessible with `d8 k`.

Deckhouse Platform (DP) is installed on top of the existing Kubernetes cluster in existing-cluster mode. This example uses Deckhouse Platform Open, the `EarlyAccess` release channel, and the `Managed` bundle.

In this setup:

- Talos continues to manage the operating system, MachineConfig, kubelet, containerd, etcd, the control plane, Kubernetes PKI, and Kubernetes updates.
- The existing CNI continues to provide Pod networking.
- The external provisioner or the user continues to create and delete machines.
- DP installs and updates platform modules but does not manage Talos or the node lifecycle.

⚠ The `bundle` value is selected during installation and cannot be changed afterwards. You cannot install `Managed` and then switch it to `Minimal` or `Default` with a regular patch.

Prerequisites

The following tools and access are required on the computer from which the installation will be performed:

- Docker
- Deckhouse CLI
- `yq` for validating YAML
- An administrative kubeconfig for the Talos cluster
- Access to the Kubernetes API
- HTTPS access to `registry.deckhouse.io` from both the computer and the cluster nodes
- `talosctl` and talosconfig if an administrative kubeconfig has not yet been obtained

SSH access to Talos nodes is not required: the installer communicates with the cluster through the Kubernetes API.

Before installation, it is recommended to create an etcd snapshot using Talos and save the original talosconfig and kubeconfig.

Setting the working paths

Create a separate directory for the installation files and change to it:

```
mkdir -p "$PWD/talos-deckhouse-install"
cd "$PWD/talos-deckhouse-install"
```

All subsequent commands assume that this remains the current directory. Set the paths once:

```
TALOSCONFIG="$PWD/talosconfig"
ADMIN_KUBECONFIG="$PWD/kubeconfig-admin"
INSTALLER_KUBECONFIG="$PWD/kubeconfig-installer"
CONFIG_FILE="$PWD/config.yml"
```

The files are used as follows:

Variable	Purpose
`TALOSCONFIG`	`talosctl` configuration for accessing the Talos API
`ADMIN_KUBECONFIG`	Administrative kubeconfig for running `d8 k` on the computer
`INSTALLER_KUBECONFIG`	Portable copy of the administrative kubeconfig for the Docker container
`CONFIG_FILE`	Deckhouse Platform installation configuration

If you open a new terminal, return to the working directory and define the four variables from the block above.

Preparing an administrative kubeconfig

The next steps depend on whether you already have an administrative kubeconfig.

If you already have a kubeconfig

Copy it to the working directory:

```
cp <ADMIN_KUBECONFIG_PATH> "$ADMIN_KUBECONFIG"
chmod 600 "$ADMIN_KUBECONFIG"
```

> **i** This refers to a kubeconfig for `d8 k`, not a talosconfig file for `talosctl`.

If you need to obtain a kubeconfig through Talos

First, copy the existing talosconfig to the working directory:

```
cp <TALOSCONFIG_PATH> "$TALOSCONFIG"
chmod 600 "$TALOSCONFIG"
```

Specify the address of a control-plane node:

```
CONTROL_PLANE_ADDRESS=<CONTROL_PLANE_IP_OR_DNS>
```

Obtain an administrative kubeconfig:

```
talosctl kubeconfig "$ADMIN_KUBECONFIG" --talosconfig="$TALOSCONFIG" --
nodes="$CONTROL_PLANE_ADDRESS" --merge=false
chmod 600 "$ADMIN_KUBECONFIG"
```

By default, `talosctl` uses the Talos API endpoints from the current talosconfig context. If a different endpoint is required, add:

```
--endpoints=<TALOS_API_ENDPOINT>
```

Use `--force` only when you intentionally want to overwrite an existing file.

Verifying permissions

Check how Kubernetes identifies the user:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" auth whoami
```

Installation requires stable administrative access. A Talos administrative kubeconfig usually uses the `system:masters` group. Using an OIDC user kubeconfig for the installation is not recommended: after the DP `user-authz` module is enabled, that user's access to system namespaces may change.

Verify the required permissions:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" auth can-i '*' '*' --all-namespaces
d8 k --kubeconfig="$ADMIN_KUBECONFIG" auth can-i create
customresourcedefinitions.apiextensions.k8s.io
d8 k --kubeconfig="$ADMIN_KUBECONFIG" auth can-i create clusterroles.rbac.authorization.k8s.io
```

All three commands must return `yes`.

Verifying the existing cluster

Check the nodes:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get nodes -o wide
```

All nodes must be in the `Ready` state.

Check the Kubernetes API:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get --raw='/readyz?verbose'
```

The response must end with `readyz check passed`.

Check the system Pods:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n kube-system get pods -o wide
```

Before installing DP, the following components must already be running:

- CNI
- CoreDNS
- kube-proxy, if it is used by the selected network setup
- Control-plane components

Also make sure that the Kubernetes version is supported by the selected DP version.

Verifying component ownership

DP must not manage the same components as Talos or an external provisioner.

The table below lists the components and their owners after installation:

Component	Owner after installation
Talos OS and MachineConfig	Talos
etcd and the Kubernetes control plane	Talos
Kubernetes PKI	Talos
kubelet and containerd	Talos
CNI	The existing external CNI
CoreDNS and kube-proxy, if used	The existing cluster
Machine creation and deletion	The external provisioner or the user
Platform modules	DP

The following DP modules must remain disabled:

- `control-plane-manager`
- `node-manager`
- `terraform-manager`
- `cni-cilium`
- `kube-dns`
- `kube-proxy`
- `cloud-provider-*` modules
- `registry-packages-proxy`

⚠ If Cilium is already installed in the Talos cluster, do not enable the DP `cni-cilium` module: two operators must not manage the same CNI at the same time.

The `Managed` bundle includes `ingress-nginx`, `cert-manager`, `local-path-provisioner`, VPA, monitoring, and the `user-authz` module. Before installation, check whether external equivalents are already present in the cluster:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get storageclass
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get ingressclass
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get deployments -A
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get crd
```

If a component is already installed, choose a single owner before proceeding. Do not run two ingress controllers, two `cert-manager` installations, or two VPA installations at the same time. If an external solution remains responsible for the component, explicitly disable the corresponding DP module using a

ModuleConfig with `spec.enabled: false`. If DP is to manage the component, disable or remove the external counterpart before installation.

Preparing a kubeconfig for the installer container

The installer runs inside Docker. It requires a portable kubeconfig that does not reference certificate and key files available only on the user's computer.

Create a portable copy:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" config view --raw --flatten --minify >
"$INSTALLER_KUBECONFIG"
chmod 600 "$INSTALLER_KUBECONFIG"
```

This command does not create new certificates. The `--flatten` option reads the CA, client certificate, and key from the paths specified in the source kubeconfig and embeds them in the new file.

Verify the copy:

```
d8 k --kubeconfig="$INSTALLER_KUBECONFIG" auth whoami
d8 k --kubeconfig="$INSTALLER_KUBECONFIG" auth can-i '*' '*' --all-namespaces
```

The second command must return `yes`.

Check the Kubernetes API address:

```
d8 k --kubeconfig="$INSTALLER_KUBECONFIG" config view --minify -o
jsonpath='{.clusters[0].cluster.server}{"\n"}'
```

This address must be reachable from the Docker container. A reachable Kubernetes API address, a VPN address, or a load balancer address is preferred.

If the address is `https://127.0.0.1:6443`, the installer cannot use it directly: inside the container, `127.0.0.1` refers to the container itself. Make the Kubernetes API accessible from the container before continuing with the installation.

Creating the configuration file

Create the `$CONFIG_FILE` file with the following content and replace `example.com` with your domain:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    bundle: Managed
    releaseChannel: EarlyAccess
    logLevel: Info
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings:
    modules:
      publicDomainTemplate: "%s.example.com"
```

The domain in `publicDomainTemplate` must not match or be a subdomain of the domain specified in `clusterDomain`. Before using the template, configure DNS services in both the networks where cluster nodes are located and the networks from which clients access the platform service web interfaces.

If the nodes have custom taints and DP components must run on them, add the corresponding values to `global.spec.settings.modules.placement.customTolerationKeys`. Do not add an example taint unless it exists in the cluster.

Validate the file:

```
yq eval-all '.' "$CONFIG_FILE" >/dev/null && echo "YAML OK"
grep -n $'\t' "$CONFIG_FILE"
```

The first command must print `YAML OK`; the second command must not print anything.

Running the Deckhouse Platform Open installer

The installer tag must match the `releaseChannel` in the configuration. The `early-access` tag is used for `EarlyAccess`.

Check that the files exist:

```
ls -l "$CONFIG_FILE" "$INSTALLER_KUBECONFIG"
```

Run the installer:

```
docker run --pull=always -it -v "$CONFIG_FILE:/config.yml:ro" -v
"$INSTALLER_KUBECONFIG:/kubeconfig:ro" registry.deckhouse.io/deckhouse/ce/install:early-access
bash
```

The installer image path still uses the `ce` designation for Deckhouse Platform Open.

Inside the installer container, run:

```
dhctl bootstrap-phase install-deckhouse --kubeconfig=/kubeconfig --config=/config.yml
```

Do not close the terminal until the bootstrap process completes. Installation can take anywhere from 5 to 30 minutes.

Monitoring the installation

In a separate terminal, change to the same working directory and define the variables from the “Setting the working paths” section again. Run the command:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n d8-system get deployment,replicaset,pods -w
```

If the `deckhouse` Pod is not created, check the events:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n d8-system get events --sort-
by=.metadata.creationTimestamp
```

Errors such as `ImagePullBackOff`, `ErrImagePull`, `401 Unauthorized`, or `403 Forbidden` usually indicate a problem with the registry address, registry access, DNS, or routing.

To diagnose a specific Pod, run:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n <NAMESPACE> describe pod <POD_NAME>
```

Verifying the installation

Wait for the main Deployment to become ready:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n d8-system rollout status deployment/deckhouse --
timeout=10m
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n d8-system get deployment,pods -o wide
```

Check the modules:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get modules -o wide
```

Enabled modules are expected to have `PHASE: Ready`, `ENABLED: True`, and `READY: True`.

The Module status alone is not sufficient: a module may be `Ready` even if one of its Deployments, StatefulSets, or DaemonSets was not created or is restarting. Check the actual resources:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get deployment,statefulset,daemonset -A
```

For each DaemonSet, the `DESIRED`, `CURRENT`, and `READY` values must match. For Deployments and StatefulSets, the expected number of replicas must be ready.

Find Pods that are not in the `Running` or `Succeeded` phase:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get pods -A --field-
selector='status.phase!=Running,status.phase!=Succeeded'
```

Check recent warnings:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get events -A --field-selector=type=Warning --sort-
by=.metadata.creationTimestamp
```

An old warning does not necessarily indicate a current problem. Consider the event timestamp, repetition count, and the current state of the related resource.

Verifying that DP does not manage Talos components

Verify that DP modules that can manage Talos components remain disabled.

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get modules control-plane-manager node-manager terraform-
manager cni-cilium kube-dns kube-proxy registry-packages-proxy -o wide
```

All listed modules must have `ENABLED: False`.

Check the cloud provider modules:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get modules -o wide | grep -E '(^NAME|^cloud-
provider-)'
```

All `cloud-provider-*` modules found by the command must have `ENABLED: False`.

Check the original cluster components again:

```
d8 k --kubeconfig="$ADMIN_KUBECONFIG" -n kube-system get pods -o wide
d8 k --kubeconfig="$ADMIN_KUBECONFIG" get nodes -o wide
```

All Talos nodes must remain `Ready`. The original CNI, CoreDNS, and control-plane components must continue to run; kube-proxy must also continue to run if it was used before the DP installation.

Successful installation criteria

The installation is considered successful when all of the following conditions are met:

- All Talos nodes remain `Ready`.
- The original CNI and CoreDNS continue to run; kube-proxy also continues to run if it was used before the DP installation.
- Deployment `deckhouse` is ready.
- Enabled modules have `READY: True`.
- The actual module Deployments, StatefulSets, and DaemonSets are ready.
- DP modules that can manage Talos components remain disabled.
- Administrative access through the Talos administrative kubeconfig is preserved.

Additional resources

- Installing Deckhouse Platform in an existing cluster

- Module `deckhouse` parameters
- Bundles and module management
- Patching Talos MachineConfig
- Global parameters
- Module `control-plane-manager` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Migrating VMs from VMware to DP

This guide describes how to migrate an existing virtual machine from VMware to Deckhouse Platform (DP).

The migration source can be:

- a virtual machine distribution (an `OVA` file, a tar archive with disk files in `vmdk` format, VM metadata in `ovf` format, and checksums in `mf` format);
- standalone `VMDK` disk files.

Migration approaches

Direct VMDK import

DP supports importing disks in `vmdk` format. You can upload a `VMDK` file from an OVA or vSphere export to a `VirtualImage` or `ClusterVirtualImage`, create a disk from the image, and then a virtual machine. See Images for the upload procedure.

> ⚠ The platform imports the disk file as-is and does not adapt the guest OS for KVM. For disks from VMware this often causes problems: the VM fails to boot, cannot see the disk, or has no network (especially on Windows). Direct import is suitable only when the `VMDK` is already prepared for QEMU/KVM.

Recommended path

For typical VMs from VMware, use the `virt-v2v` utility: it converts `VMDK` to `qcow2` and adapts the guest OS for KVM (virtio drivers, bootloader, replacement of VMware devices). Upload the prepared disk directly to a `VirtualDisk` with `type: Upload`. The volume is provisioned in the chosen StorageClass, bypassing DVCR.

The step-by-step instructions below cover this scenario.

Migration stages

Migrating a VM from VMware to DP includes the following stages:

1. Install the required tools.
2. Convert the disk.
3. Upload the disk to the cluster as a VirtualDisk resource.
4. Create a virtual machine (VirtualMachine resource) that boots from this disk.

What you need for migration

Before you start the migration, make sure you have:

- access to the DP cluster with the Deckhouse CLI utility (`d8`) installed and permissions to create virtualization resources in the target namespace;
- a Linux host with `virt-v2v` and `libguestfs` installed and enough disk space to unpack the `OVA` (or standalone `VMDK` files) and store the conversion output;
- the VMware export files (`OVA` or `VMDK`).

For more details on uploading disks to the cluster, see Disks.

Install tools

On this step you prepare a conversion workstation. It does not have to be a DP cluster node: any Linux host with internet access or a local package repository is sufficient.

The packages you need depend on the guest OS in the VM you are migrating:

- for Linux, `virt-v2v` and `libguestfs` are sufficient;
- for Windows, you will additionally need a `virtio-win` ISO so the guest OS works correctly with virtual devices in KVM after migration.

Install the tools:

- Ubuntu/Debian workstation
- RHEL/AlmaLinux workstation

Run the following command:

```
sudo apt update
sudo apt install -y virt-v2v libguestfs-tools
```

If the VM being migrated runs Windows:

1. Download the VirtIO drivers from the `virtio-win` distribution.
2. Specify the path to the VirtIO drivers via an environment variable:

   ```
   export VIRTIO_WIN=/path/to/virtio-win.iso
   ```

⚠ Without a valid `virtio-win` ISO for a Windows guest, conversion may fail, or the guest OS may not see disks or networking after the VM starts in DP.

Run the following command:

```
sudo dnf install -y virt-v2v libguestfs-tools-c virtio-win
```

If the VM being migrated runs Windows, set the path to the ISO via an environment variable:

```
export VIRTIO_WIN=/path/to/virtio-win.iso
```

Then proceed to converting the disk.

Convert the disk

On this step you convert VMware data into one or more `qcow2` files that DP can use as virtual machine volumes. If you already have a ready `VMDK`, go straight to Convert VMDK to qcow2 via virt-v2v. If you are using a virtual machine `OVA` distribution, unpack it first.

Extract an OVA

An `OVA` file is a tar archive with a manifest, an `OVF` descriptor, and one or more `VMDK` images. Unpacking gives you the disk file path for `virt-v2v`. Keep the `OVF` file: you will use it later for CPU, memory, and bootloader settings in the VirtualMachine resource.

Extract everything if you want to verify checksums or inspect the OVF:

```
tar -xvf machine.ova
```

Typical contents:

```
machine.ova
├── machine.mf          # checksums (SHA256)
├── machine.ovf         # VM metadata (CPU, RAM, disks, networks)
└── machine-disk1.vmdk  # disk image
```

If the archive is large, you can extract only the required `VMDK` listed in the `OVF`:

```
tar -xvf machine.ova machine-disk1.vmdk
```

> **i** Virtual machines with multiple disks contain several `*.vmdk` files. Convert each disk with the `virt-v2v` utility, create a matching `VirtualDisk` in DP, then reference them from `VirtualMachine` in the desired boot order.

Convert VMDK to qcow2 via virt-v2v

With `-i disk`, `virt-v2v` processes a local `VMDK` and saves the result into the directory you specify. To perform the conversion, run:

- For a Linux guest OS
- For a Windows guest OS

```
virt-v2v -i disk ./machine-disk1.vmdk \
    -o local -os ./out -of qcow2
```

To convert a `VMDK` for a Windows guest, specify the path to `virtio-win.iso` in the command:

```
VIRTIO_WIN=/path/to/virtio-win.iso virt-v2v -i disk ./machine-disk1.vmdk \
    -o local -os ./out -of qcow2
```

After the conversion, a file such as `./out/machine.qcow2` will appear under `./out` (the basename often matches the original VM name from the metadata). That file is what you upload to the cluster next.

Upload the disk to the cluster

This section describes how to transfer the prepared `qcow2` image to DP through the Kubernetes API. At this stage the file becomes a persistent volume in the cluster: create a VirtualDisk with `type: Upload` and transfer the `qcow2` over HTTP. The disk is provisioned in the chosen StorageClass, bypassing DVCR.

Uploading the disk image to the cluster includes the following steps:

1. Choose a StorageClass.
2. Create a VirtualDisk for upload.
3. Get upload URLs.
4. Upload the image.
5. Check the status of the uploaded image.

Choose a StorageClass

StorageClass in Kubernetes defines where and how the volume is provisioned; in VMware terms this is closest to a `datastore`. Performance, replication type, and volume expansion policy depend on the class.

List classes available in your cluster:

```
d8 k get storageclass
```

Example:

```
NAME                 PROVISIONER                          VOLUMEBINDINGMODE   AGE
rv-thin-r1 (default) replicated.csi.storage.deckhouse.io  Immediate           48d
rv-thin-r2           replicated.csi.storage.deckhouse.io  Immediate           48d
```

Select the class name that fits your storage requirements for VM disks.

Create a VirtualDisk for upload

Create the disk resource, specifying StorageClass and volume size.
`spec.persistentVolumeClaim.size` must be at least the actual size of the `qcow2` you upload. If unsure, leave margin: if the size is insufficient, recreate the resource with a larger PVC.

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: uploaded-disk
spec:
  persistentVolumeClaim:
    storageClassName: rv-thin-r1
    size: 10Gi
  dataSource:
    type: Upload
EOF
```

After creation the resource moves to `WaitForUserUpload`: the volume is allocated and you can start transferring the file.

Get upload URLs

The platform provides two URLs: internal (`imageUploadURLs.inCluster`) and external (`imageUploadURLs.external`). Use the address reachable from your network (from inside the cluster or from an administrator workstation).

Internal URL (use when uploading from a cluster node or from a pod):

```
d8 k get vd uploaded-disk -o jsonpath="{.status.imageUploadURLs.inCluster}"
```

External URL (use from an administrator workstation when access to DP is configured):

```
d8 k get vd uploaded-disk -o jsonpath="{.status.imageUploadURLs.external}"
```

Both fields together (requires `jq`):

```
d8 k get vd uploaded-disk -o jsonpath="{.status.imageUploadURLs}" | jq
```

⚠ The URL string contains a secret path segment. Do not publish it in public channels.

Upload the image

Send the `qcow2` image with an HTTP `PUT` request to the URL obtained in the previous step. The example below uses an external URL. Replace it with the address from your `VirtualDisk` status and the path to the converted file.

```
curl https://virtualization.example.com/upload/<secret-url> \
    --progress-bar -T ./out/machine.qcow2 | cat
```

Wait until the transfer completes successfully without HTTP errors. The controller will process the image and transition the disk to `Ready`.

Check status

Verify that the disk resource is healthy and the volume size looks correct:

```
d8 k get vd uploaded-disk
```

Example:

```
NAMESPACE   NAME            PHASE   CAPACITY   AGE
default     uploaded-disk   Ready   10Gi       1m
```

If the phase stays on `WaitForUserUpload` for a long time or the resource enters `Failed`, check messages with `d8 k describe vd uploaded-disk` and events in the corresponding namespace.

Continue when the disk reaches `Ready`.

Create the virtual machine

The final step is to describe the VM that will boot from the migrated disk. Specify how much CPU and memory to allocate, which network to attach, and which disk is bootable. VMware configuration (`OVF`/`VMX`) is not imported directly: carry over parameters manually using the `OVF` file and the mapping table below.

VMware vs DP terminology

If you know vSphere, the following table maps familiar VMware objects to Kubernetes and DP virtualization resources.

VMware	DP	Description
Datastore	StorageClass	Disk backing storage
VMX	VirtualMachine.spec	VM specification
Virtual disk (VMDK)	VirtualDisk	VM disk
ISO image	VirtualImage (`cdrom: true`)	Installation or driver ISO
Template	VirtualImage	Template for provisioning disks
Port group / VLAN	VirtualMachine (`networks`)	Networking
Resource pool	Project and quotas	Resource limits per project
Snapshot	VirtualDiskSnapshot / VirtualMachineSnapshot	Disk and VM snapshots
Folder	Namespace	Namespace
Cluster / resource pool	Project	Namespace grouping
ESXi host	Node	Physical server
vCenter	Kubernetes API	Cluster management

For more details on connecting VMs to networks, see Virtual machine networks.

VirtualMachine example

The VirtualMachine resource references the uploaded disk through `blockDeviceRefs`. Order in `blockDeviceRefs` determines boot order: the boot disk must be listed first.

Minimal Linux example after disk migration:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: my-vm
spec:
  virtualMachineClassName: generic
  osType: Generic
  cpu:
    cores: 2
  memory:
    size: 4Gi
  networks:
    - type: Main
  blockDeviceRefs:
    - kind: VirtualDisk
      name: uploaded-disk
EOF
```

Set `osType: Windows` for Windows guests.

If the source VM in VMware used UEFI boot, add `bootloader: EFI` (see the parameter table below).

If additional networks and the SDN module are configured in the cluster, you can add interfaces alongside the primary network:

```
networks:
  - type: Main
  - type: Network
    name: user-net
```

Additional capabilities (cloud-init, multiple disks, VM classes for production environments) are described in Virtual machines.

Common spec fields

The following table lists fields you most often need to verify after migration from VMware. CPU, memory, and bootloader values can usually be taken from the unpacked `OVF` file.

Field	Description
`virtualMachineClassName`	VM class, e.g. `generic`, `serverful`, `high-performance`
`osType`	`Generic` (Linux and others) or `Windows`
`bootloader`	Boot firmware: `BIOS`, `EFI`, or `EFIWithSecureBoot`; for UEFI VMs in VMware, set `EFI`
`cpu.cores`	Number of vCPUs
`memory.size`	RAM
`blockDeviceRefs`	Disks and images; list order is boot order
`provisioning.type: UserData`	cloud-init user data for first boot of the guest OS

Check VM status

After applying the manifest, wait until the VM is running and has an address (when the primary network assigns IPs from `virtualMachineCIDRs`):

```
d8 k get vm my-vm
```

Example:

```
NAME    PHASE     NODE           IPADDRESS     AGE
my-vm   Running   virtlab-pt-2   10.66.10.12   2m
```

If the phase is `Pending` or startup fails, use `d8 k describe vm my-vm`, the serial console `d8 v console my-vm` (see Connect to the VM below), and review virtualization component logs in the cluster.

Connect to the VM

Choose an access method depending on whether the guest OS accepts SSH, you need a graphical console, or serial console is enough for boot troubleshooting.

Method	Purpose	Command
Serial console	Bootloader and kernel output	`d8 v console my-vm`
VNC	Graphical console without SSH	`d8 v vnc my-vm`
SSH	Remote shell access to the guest	`d8 v ssh cloud@my-vm --local-ssh`

The SSH username is defined in the guest OS; examples in the DP virtualization documentation often create a `cloud` user via cloud-init.

Additional resources

- Module `virtualization` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Architecture

This section describes the architecture of Deckhouse Platform (DP).

The section consists of the following subsections:

- C4 model: Overview of the C4 model used to visualize the platform architecture, as well as a description of the DP architecture at levels 1 and 2 of the C4 model.
- Modules: Description of the DP module architecture.
- Disaster resilience: Description of the disaster resilience approaches implemented in DP.
- Updating: Description of the DP update mechanisms.
- Description of platform component architecture divided in the following subsystems:
  - Deckhouse subsystem
  - Kubernetes & Scheduling subsystem
  - Cluster & Infrastructure subsystem
  - IAM subsystem
  - Security subsystem
  - Network subsystem
  - Storage subsystem
  - Observability subsystem

> This section does not yet cover all DP subsystems and modules.
> Documentation for the remaining components will be added as it becomes available.

DP architecture

DP is a platform for managing Kubernetes clusters in various infrastructures — from isolated server environments to public clouds. The platform includes:

- A Kubernetes cluster.
- The Deckhouse controller and the modules it manages.
- Bashible, an agent running as a service on cluster nodes that executes bash scripts to manage nodes.

Modules are grouped into subsystems according to their functional purpose. The Deckhouse controller is also implemented as a module and is the only mandatory module required for the platform to function.

The DP architecture at the subsystem and module level is described in the C4 model subsection.

Modules

A module is a set of resources and applications designed to extend DP functionality.

Key modules:

- `deckhouse`: The Deckhouse controller.
- `control-plane-manager`: Manages cluster control plane components.
- `node-manager`: Manages cluster nodes.

> The `control-plane-manager` and `node-manager` modules are not installed when DP is installed into an existing Managed Kubernetes cluster.

Module contents:

- Helm charts
- Addon-operator hooks
- Build rules for module components (Deckhouse components)

- Other related files

DP uses the addon-operator project to manage modules. Refer to its documentation to learn how DP works with modules, module hooks, and module parameters.

For more information about module architecture and developing custom modules, refer to Modules.

Additional resources

- Module `control-plane-manager` documentation
- Module `deckhouse` documentation
- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

DP architecture (C4 model)

The C4 model is used to visually represent a system's architecture and explain how it is decomposed into elements. Diagrams in this model are organized according to increasing levels of scale and detail:

- **Level 1: Context diagram.** Shows the system within its external context: user roles and other systems that directly interact with it.
- **Level 2: Container diagram.** Breaks the system down into containers (major subsystems) and shows the main interaction paths between them. In the C4 model, a container (not to be confused with Docker containers) is an executable and deployable subsystem, such as an executable file.
- **Level 3: Component diagram.** Breaks each container down into components and shows interactions between components (within a single container) and with other containers. In the C4 model, a component is a software module (subsystem) within the container's codebase. This level is described in the architectural diagrams of individual services.
- **Level 4: Code diagram.** Describes the code elements that implement the components.

Due to the large number of elements at Level 2, the complete container diagram is not shown here. Instead, the following intermediate diagrams are provided after the Level 1 diagram:

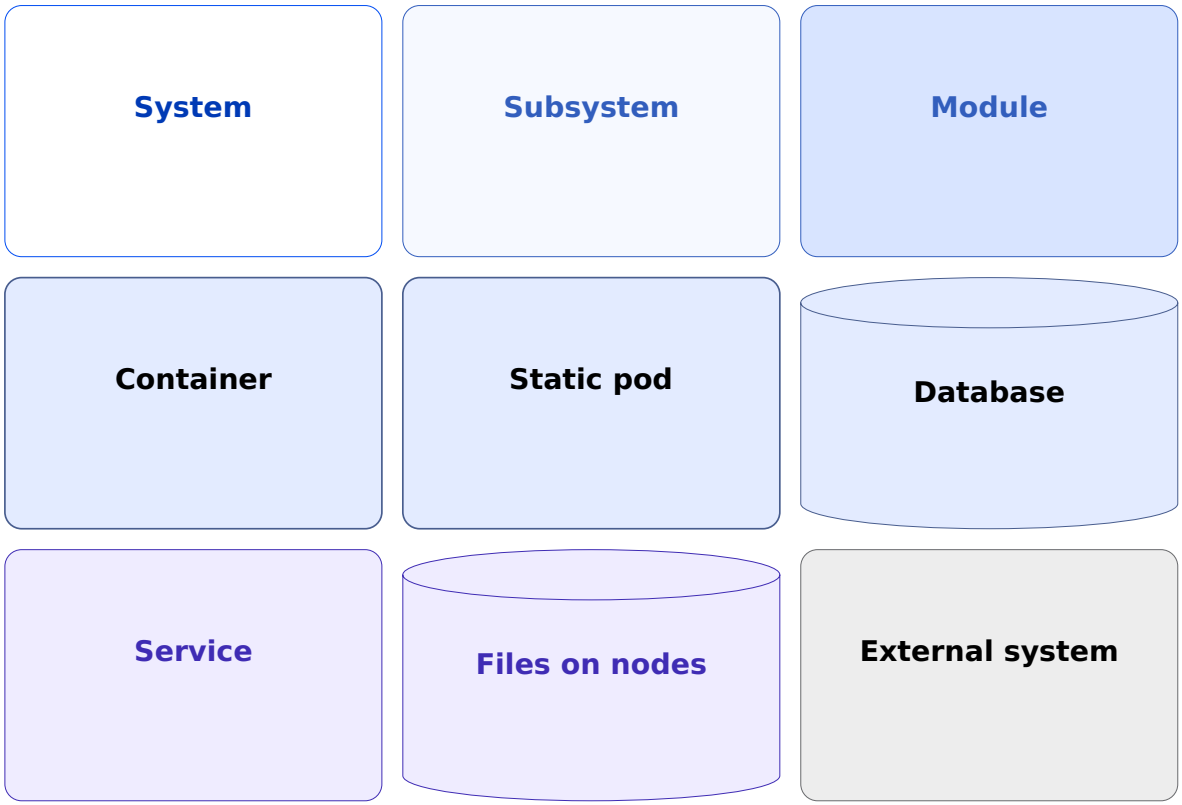
- Subsystem diagram
- Module diagram

Level 2 and higher diagrams are provided in the sections describing the architecture of individual modules.

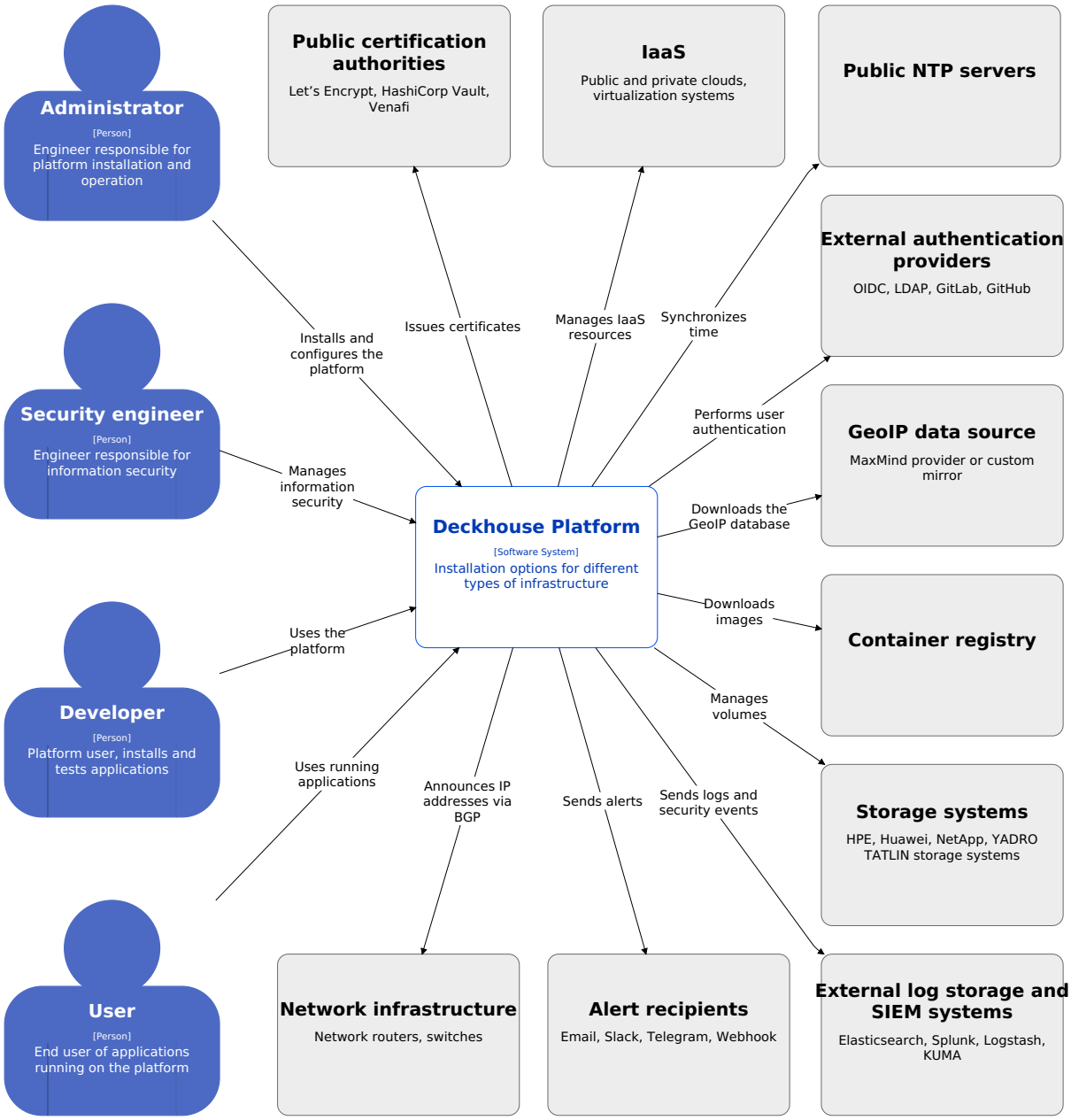
To get a general understanding of the Deckhouse Platform (DP) architecture, it is sufficient to review the Level 1 diagram as well as the subsystem and module diagrams. For a detailed view of individual services, refer to the Level 2 and higher diagrams available in the relevant sections.

Legend

The following graphical symbols are used in the diagrams:



Context diagram



The following roles interact with DP:

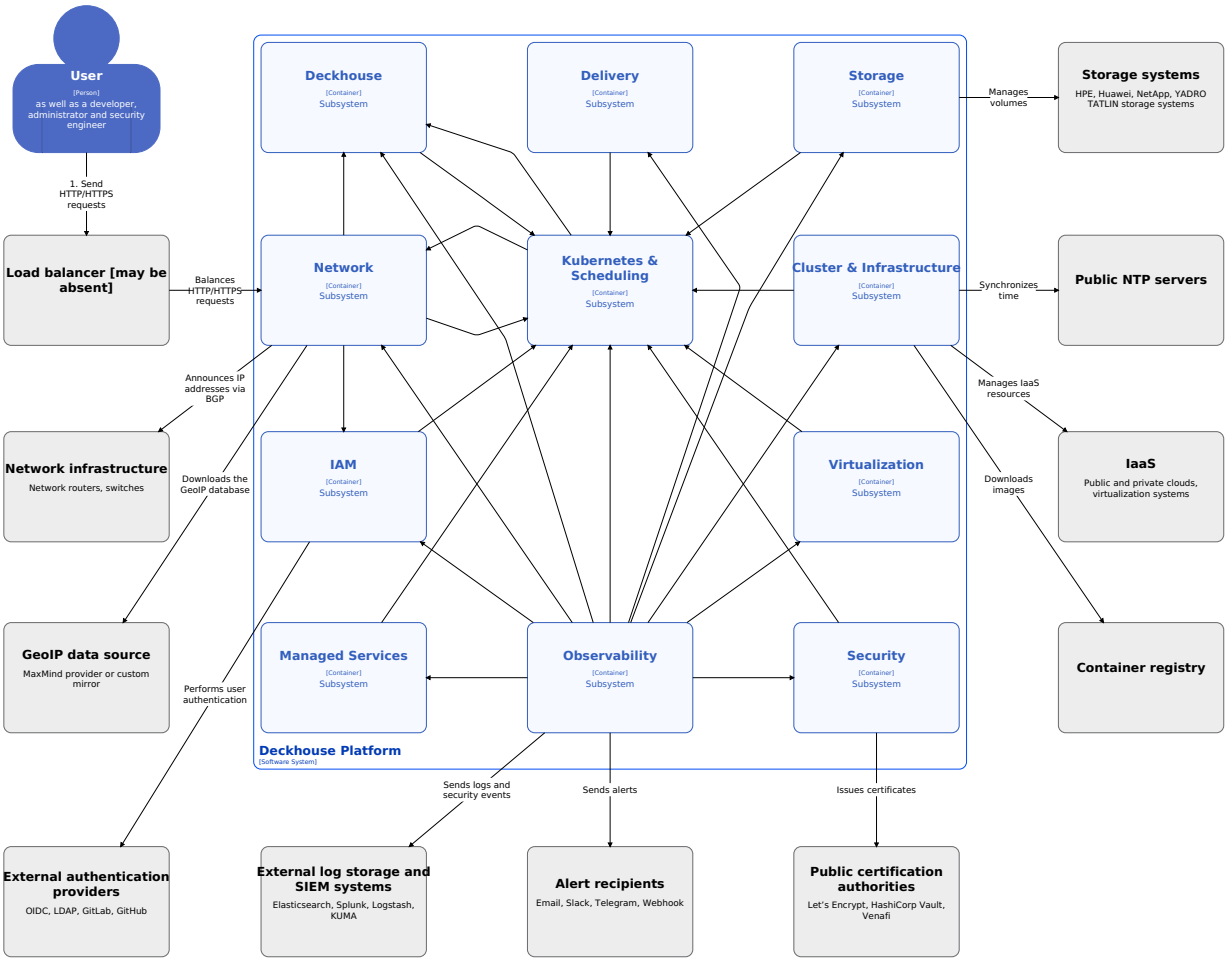
- **Administrator**: Installs and configures the platform.
- **Security Engineer**: Manages information security.
- **Developer**: Uses the platform to deploy and test applications under development.
- **User**: Uses applications running on the platform.

Interactions with external services are described in the diagrams of subsequent levels.

Subsystem diagram

The diagram below shows the DP subsystems.

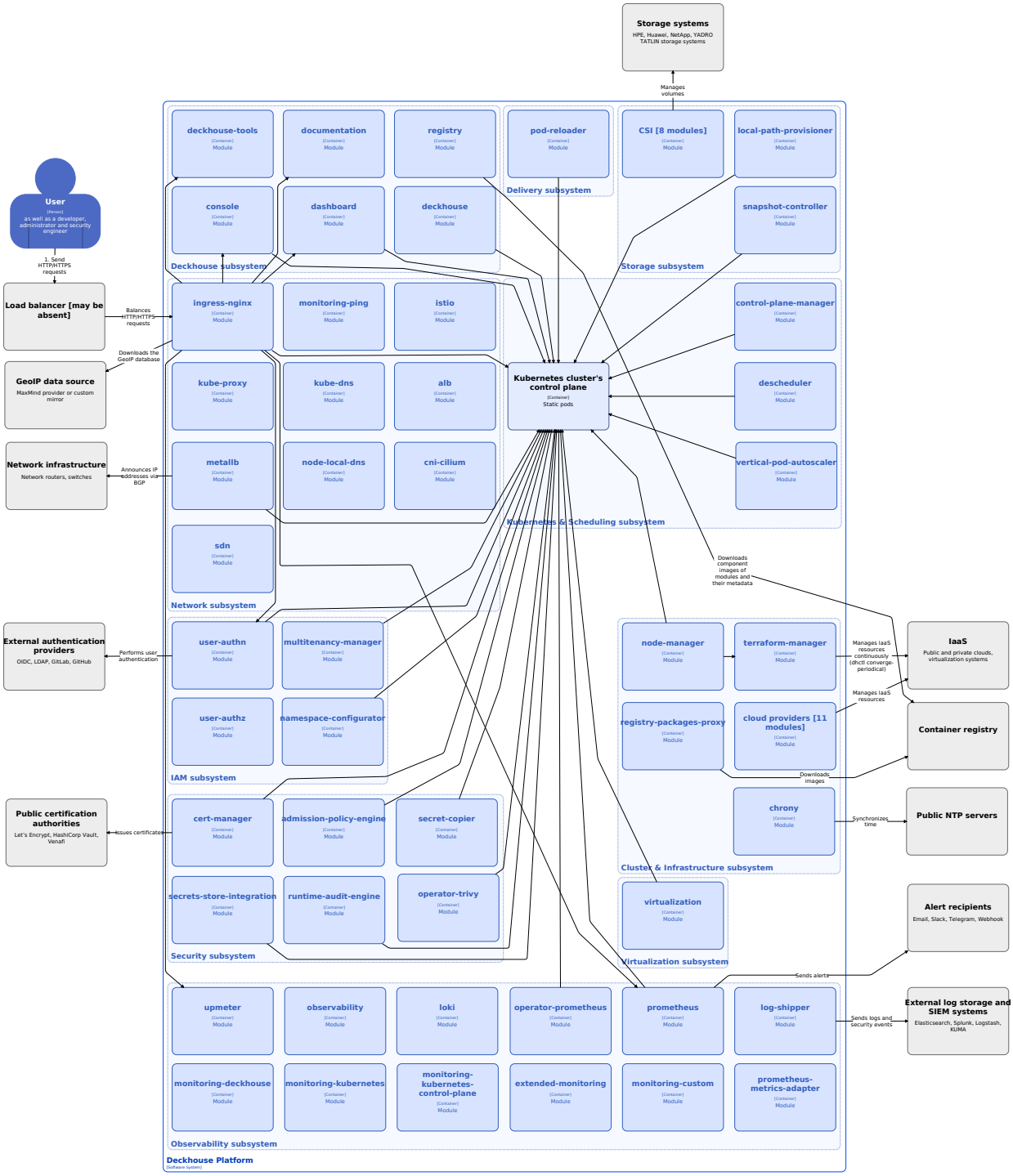
> **i** Relationships between subsystems are shown in a generalized form and without detailed interactions.



Module diagram

The diagram below shows all modules included in the Default module bundle, as well as the most important additional modules. Modules are grouped by subsystem.

> **i** Relationships between modules are shown in a generalized form and without detailed interactions.



Additional resources

- The C4 model for visualising software architecture

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse Platform module development

A module is a set of resources and applications designed to extend the functionality of the Deckhouse Platform.

This section is dedicated to the architecture of DP modules. It will help you understand how modules are structured, how to develop and debug them, and how they interact with other components of the

platform.

Deckhouse Platform (DP) can work with the following types of modules:

- Built-in modules. These are included in DP. Their release cycle is tied to the DP release cycle.
- Modules from module source. The release cycle for these modules is not tied to the DP release cycle.

Creating a module consists of the following stages:

- **Development**: creating module code and its structure in the Git repository. The **Module structure** section outlines which components there are and in which directories they are located.
- **Building**: creating a module artifact and pushing it to the container registry. The **Building and publishing** section describes where images are stored in the registry and at what paths they are available.
- **Running in a cluster**: delivering the module to a cluster managed by the DP. The **Running in a cluster** section describes how to activate the module, configure its parameters, and test its functionality (including handling CRDs and troubleshooting).
- **Dependencies**: configuring module dependencies, including DP versions, Kubernetes, and other critical components. This stage is covered in the **Module dependencies** section.

Requirements

You will need the following tools to develop DP modules:

- git — version control system;
- sed — stream editor;
- yq — CLI tool for processing data in JSON, YAML, and XML formats;
- jq — CLI tool for processing data in JSON, YAML, and XML formats;
- werf — (optional) CLI tool for building images. You will need it for building module artifacts locally;
- crane — (optional) CLI tool for working with the container registry. You may need it when debugging.

The container registry where module artifacts will be stored must support a nested repository structure. A registry such as Docker Registry v2 or Harbor is a good choice.

Before you start

To get an idea of how DP modules work, check out addon-operator and shell-operator.

- Review the operator documentation on the concept of hooks, e.g., what a hook configuration is and what functions it provides. The configuration is used to configure the data that will be available from the hook.
- Check out bindings. Bindings are events that trigger the hook. They are specified in the hook configuration. A hook can be triggered not only by Kubernetes events, but also, e.g., on a schedule or before a module is started.

  > **i** The Hook allows you to keep values in memory and use them later when rendering Helm templates. We recommend reading the Hooks and Helm values section to learn more about this feature as well as the module's operating cycle.

- Explore the concept of snapshots. With snapshots, you can implement a reconciliation loop approach that is more efficient than event subscription.

  > **i** This is how DP implements support for all existing backend module hooks.

- Additionally, hooks can be used instead of the Prometheus exporter. Hooks can provide metrics that DP will export. See metrics.

Got a question?

The Development and debugging section contains information about what tools can be used in module development and approaches to troubleshooting module errors.

Join the community, where you'll be sure to have your questions answered.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module structure

The source code of the module and its assembly rules must be located in a directory with a certain structure. The best analog is a Helm chart.

Not all module folders and files are required. In short, you can use the following guidelines:

- Create a module.yaml file to describe the module metadata.
- In the templates folder, place the Helm templates that will be used in the cluster.

  If you want the objects created by the module to change their behavior depending on any module parameters, define the necessary parameters in the specification and use them in the templates.
- In the images folder, place instructions for building container images used by the module.

  If the templates only use addresses of *external* images, you don’t need to create a folder.
- If the module should respond to events or interact with the Kubernetes API, you need to create hooks that should be placed in the hooks folder.
- Place the module documentation in the docs folder. If the module documentation is missing, the module will not appear in the list of modules in the documentation web interface in the cluster.
- If the module has auxiliary Helm charts, place them in the charts folder.
- If the module needs to create custom resources (CRD), place their specifications in the crds folder.

There is a repository containing the sample module template. We recommend you start your module development with it.

Further down this page you will find a more detailed description of the module directory and file structure.

▼ Example module folder structure...

Example module folder structure containing build and publish rules using GitHub Actions:

```
 my-module/
├──  .github/
│   ├──  workflows/
│   │   ├──  build_dev.yaml
│   │   ├──  build_prod.yaml
│   │   ├──  checks.yaml
│   │   ├──  deploy_dev.yaml
│   │   └──  deploy_prod.yaml
├──  .werf/
│   ├──  workflows/
│   │   ├──  base-images.yaml
│   │   ├──  bundle.yaml
│   │   ├──  images.yaml
│   │   ├──  images-digest.yaml
│   │   ├──  batch-go.yaml
│   │   └──  release.yaml
├──  charts/
│   └──  helm_lib/
├──  crds/
│   ├──  crd1.yaml
│   ├──  doc-ru-crd1.yaml
│   ├──  crd2.yaml
│   └──  doc-ru-crd2.yaml
├──  docs/
│   ├──  README.md
│   ├──  README.ru.md
│   ├──  EXAMPLES.md
│   ├──  EXAMPLES.ru.md
│   ├──  CONFIGURATION.md
│   ├──  CONFIGURATION.ru.md
│   ├──  CR.md
│   ├──  CR.ru.md
│   ├──  FAQ.md
│   ├──  FAQ.ru.md
│   ├──  ADVANCED_USAGE.md
│   └──  ADVANCED_USAGE.ru.md
├──  hooks/
│   ├──  batch/
│   │   ├──  my-hooks/
│   │   │   ├──  my-hook1.go
│   │   │   └──  my-hook2.go
│   │   ├──  go.mod
│   │   ├──  go.sum
│   │   ├──  main.go
├──  images/
│   ├──  nginx
│   │   └──  Dockerfile
│   └──  backend
│       └──  werf.inc.yaml
├──  lib/
│   └──  python/
│       └──  requirements.txt
├──  openapi/
│   ├──  conversions
│   │   ├──  testdata
│   │   │   ├──  v1-1.yaml
│   │   │   └──  v2-1.yaml
│   │   ├──  conversions_test.go
│   │   └──  v2.yaml
│   ├──  config-values.yaml
│   ├──  doc-ru-config-values.yaml
│   └──  values.yaml
├──  templates/
│   ├──  a.yaml
│   └──  b.yaml
├──  .helmignore
├──  Chart.yaml
├──  module.yaml
├──  werf.yaml
└──  werf-giterminism.yaml
```

charts

The `/charts` directory contains Helm helper charts used when rendering templates.

Deckhouse Platform (DP) has its own library for working with templates called lib-helm. You can read about the library's features in the lib-helm repository. To add the library to the module, download the tgz-archive with the appropriate release and move it to the `/charts` directory of the module.

crds

This directory contains *CustomResourceDefinitions* (CRDs) used by the module components. CRDs are

updated every time the module is started, if there are updates.

Nested directories in the `/crds` folder are ignored.

> **i** If your module's resources have fields that reference cluster-wide resources (such as a StorageClass, ClusterIssuer, or ClusterRole), consider registering such fields using a GrantableClusterResourceReference. In this case, the specified cluster-wide resources will be validated per-project and can use the default value assignment. For details, refer to the `multitenancy-manager` documentation.

To render CRDs from the `/crds` directory in the site documentation or documentation module in the cluster, follow these steps:

- Create a translation file with a structure identical to the original resource file:
  - in it, keep only the `description` parameters containing the translation text;
  - use the `doc-ru-` prefix in the name: e.g., `/crds/doc-ru-crd.yaml` for `/crds/crd.yaml`.
- Create `/docs/CR.md` and `/docs/CR.ru.md` files.

docs

> **i** The module lifecycle is specified in module.yaml. The availability of the module in the editions of the Deckhouse Platform is not determined by the module developer.

The `/docs` directory contains the module documentation. The following nested directories aren't involved in the documentation building process:

- `internal`
- `internals`
- `development`
- `dev`

The following files are required for building the documentation:

- `README.md` and `README.ru.md` — this file (and its localized version) describes what the module is for, what problem it solves and outlines the general architectural principles.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` — **(recommended)** The title of the module description page, for example, "Deckhouse web admin console". It is also used in navigation if `linkTitle` parameter is not specified.
  - `menuTitle` — **(recommended)** The name of the module to show in the menu on the left sidebar of the page, e.g., "Deckhouse Admin". If not set, the name of the directory or repository is used, e.g. `deckhouse-admin`.
  - `linkTitle` — **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.
  - `description` — **(recommended)** A short unique description of the page content (up to 150 characters). It should not repeat the `title`. Goes on with the meaning of the title and reveals it in more detail. It is used during generation of preview links and indexing by search engines, e.g., "The module allows you to fully manage your Kubernetes cluster through a web interface with only mouse skills."

  ▼ **Metadata example...**

  ```
  ---
  title: "Deckhouse administrator web console"
  menuTitle: "Deckhouse Admin"
  description: "The module allows you to fully manage your Kubernetes cluster through a web interface with
  only mouse skills."
  ---
  ```

The following files aren't required but have a predetermined name in the sidebar menu and the page title:

- `EXAMPLES.md` and `EXAMPLES.ru.md` — this file (and its localized version) contains examples of module configuration with description.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page, e.g., `Examples`. It is also used in navigation if there is no `linkTitle`.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters). It should not repeat the `title`. Goes on with the meaning of the title and reveals it in more detail. It is used during generation of preview links and indexing by search engines, e.g.,

“Examples of storing secrets in a neural network and automatically substituting them into thoughts when communicating.”
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

▼ **Metadata example...**

```
---
title: "Examples"
description: "Examples of storing secrets in a neural network with automatic substitution into thoughts
when communicating."
---
```

- `FAQ.md` and `FAQ.ru.md` — this file (and its localized version) contains frequently asked questions related to module operation, e.g., “What scenario should I choose: A or B?”.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters).
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

  ▼ **Metadata example...**

  ```
  ---
  title: "FAQs"
  description: "Frequently asked questions."
  ---
  ```

- `ADVANCED_USAGE.md` and `ADVANCED_USAGE.ru.md` — this file (and its localized version) contains expanded instructions on using and debugging the module.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters).
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

  ▼ **Metadata example...**

  ```
  ---
  title: "Module debugging"
  description: "This section covers all the steps for debugging the module."
  ---
  ```

- `CR.md` and `CR.ru.md` — the files for generating resources from the `/crds/` directory. Add these files if such a generation is required.

  ▼ **Metadata example...**

  ```
  ---
  title: "Custom resources"
  ---
  ```

- `CONFIGURATION.md` and `CONFIGURATION.ru.md` — this file (and its localized version) is used to generate OpenAPI specifications from `/openapi/config-values.yaml` and `/openapi/doc-<LANG>-config-values.yaml`. Add these files if such a generation is required.

  ▼ **Metadata example...**

  ```
  ---
  title: "Module settings"
  ---
  ```

All images, PDF files and other media files should be stored in the `/docs` directory or its subdirectories (e.g, `/docs/images/`). All links to files should be relative.

You need a file with the appropriate suffix for each language, for example `image1.jpg` and `image1.ru.jpg`. Use the following links:

- `[image1](image1.jpg)` in an English-language document;
- `[image1](image1.ru.jpg)` in a Russian-language document.

hooks

The `/hooks/batch` directory contains the module's hooks. A hook is an executable file executed in response to an event. Hooks are also used by the module for dynamic interaction with Kubernetes API. For example, they can be used to handle events related to the creation or deletion of objects in a cluster.

Get to know the concept of hooks before you start developing your own hook. You can use the Go library by the Deckhouse team to speed up the development of hooks.

Hook requirements:

- When run with the `hook config` arguments, it should output its hook configuration in JSON format.
- When run with the `hook list` arguments, it should output a run of all hooks with their sequence number.
- When run with the `hook run 0` arguments, the logic of the hook with number 0 should be executed.

The hook files must be executable. Add the appropriate permissions using the `chmod +x <path to the hook file>` command.

You can find Go hook examples in the module template repository. Go hook examples can also be found in the SDK.

images

The `/images` directory contains instructions for building module container images. The first level contains directories for files used to create the container image, the second level contains the building context.

There are two ways to define a container image:

1. Dockerfile — this file contains commands for building images. To build an application from source code, copy it next to the Dockerfile and include it in the image using the `COPY` command.
2. The `werf.inc.yaml` file, which is the same as the image definition section in `werf.yaml`.

The image name matches the directory name for this module, written in *camelCase* notation starting with a small letter. For example, the directory `/images/echo-server` corresponds to the image name `echoServer`.

The built images have content-based tags that can be used when building other images. To use content-based image tags, enable the lib-helm library. You can also use other features of the helm_lib library of Deckhouse Platform.

Below is an example of using a content-based image tag in a Helm chart:

```
image: {{ include "helm_lib_module_image" (list . "<image name>") }}
```

openapi

conversions

The `/openapi/conversions` directory contains module parameter conversion files and their tests.

Module parameter conversions allow you to convert the OpenAPI specification of module parameters from one version to another. Conversions may be necessary when a parameter is renamed or moved to a different location in a new version of the OpenAPI specification.

Each conversion can only be performed between two consecutive versions (e.g., from the first to the second one). There can be several conversions, and the chain of conversions must sequentially cover all versions of the parameter specification with no "gaps".

The conversion file is a YAML file named `v<N>.yaml` or `v<N>.yml`, where `<N>` is the conversion version. It has the following structure:

```
# Version number of the module parameter specification to which the data is converted
during the conversion.
version: N
# A set of jq expressions used during conversion in the cluster for the automatic
transformation
# of the previous version module parameters.
conversions: []
# Actions (in two languages) that need to be taken to convert the data
# from the previous version of the module parameter specification.
description:
  ru: ""
  en: ""
```

The description of actions specified in the description section of the conversion file must be clear and include information on which parameters and in what order need to be changed in the module's parameter specification in order to switch to the new version.

Below is an example of a module parameter conversion file `v2.yaml`, where in version 2 the `.auth.password` parameter has been removed:

```
version: 2
conversions:
  - del(.auth.password) | if .auth == {} then del(.auth) end
description:
  ru: "Remove `.auth.password`, then `auth` if empty."
  en: "Remove `.auth.password`, then `auth` if empty."
```

Conversion tests

You can use the `conversion.TestConvert` function to write conversion tests. It receives the following parameters:

- Path to the source configuration file (i.e., the version before the conversion);
- Path to the resulting configuration file (i.e., the version after the conversion).

An example of a conversion test.

config-values.yaml

This file is required to validate the module parameters that the user can configure via ModuleConfig.

To render the schema in the documentation on the site or in the documentation module in the cluster, create:

- The `doc-ru-config-values.yaml` file with a structure similar to that of the `config-values.yaml` file. Keep only the translated description parameters in the `doc-ru-config-values.yaml` file;
- The `/docs/CONFIGURATION.md` and `/docs/CONFIGURATION.ru.md` files to enable rendering of data from the `/openapi/config-values.yaml` and `/openapi/doc-ru-config-values.yaml` files.

An example of a `/openapi/config-values.yaml` schema with a single configurable `nodeSelector` parameter:

```
type: object
properties:
  nodeSelector:
    type: object
    additionalProperties:
      type: string
    description: |
      The same as the Pods' `spec.nodeSelector` parameter in Kubernetes.

      If the parameter is omitted or `false`, `nodeSelector` will be determined
      automatically.</code>
```

An example of the `/openapi/doc-ru-config-values.yaml` file:

```
properties:
  nodeSelector:
    description: |
      English description. Markdown markup.</code>
```

Validations with x-deckhouse-validations

When developing a module for the Deckhouse Platform, you can use the OpenAPI extension `x-deckhouse-validations` to describe complex validation rules for module parameters using Common Expression Language (`CEL`).

When using these validation rules, keep the following features in mind:

- Validations can be placed at the root level or inside any property (including inside objects, arrays, and additionalProperties).
- All parameters at the current level are available in expressions via the `self` variable.
- The previous value at the same level is available in expressions via the `oldSelf` variable. Rules that

reference `oldSelf` are treated as transition rules and are evaluated only on updates (when a previous value exists at this level); on the initial create or for newly added subtrees they are silently skipped. This is the same model as `x-kubernetes-validations`.
- Validation works recursively: all nested objects, arrays, and maps can also contain their own `x-deckhouse-validations`.
- Supported types: scalars, arrays, objects, and maps (`additionalProperties`).
- If there are multiple validation errors, the user will see all messages from the corresponding rules.

Examples of rules

Below are examples of complex validation rules described in CEL:

- Checking whether the parameter value falls within the range:

```
type: object
properties:
  replicas:
    type: integer
  minReplicas:
    type: integer
  maxReplicas:
    type: integer
x-deckhouse-validations:
  - expression: "self.minReplicas <= self.replicas && self.replicas <=
self.maxReplicas"
    message: "replicas must be between minReplicas and maxReplicas"
```

- Checking for the presence of a key:

```
- expression: "'Available' in self.stateCounts"
  message: "The key 'Available' must be present"
```

- Checking that exactly one of two lists is non-empty:

```
- expression: "(self.list1.size() == 0) != (self.list2.size() == 0)"
  message: "Exactly one of the lists must be non-empty"
```

- Checking a value by regular expression:

```
- expression: "self.details.all(key, self.details[key].matches('^[a-zA-Z]*$'))"
  message: "All values must contain only letters"
```

- Marking a field as immutable using the `oldSelf` variable (transition rule):

```
type: object
properties:
  clusterDomain:
    type: string
x-deckhouse-validations:
  - expression: "self.clusterDomain == oldSelf.clusterDomain"
    message: "clusterDomain is immutable"
```

  The rule is only evaluated on updates: on the initial create (when there is no previous value) it is automatically skipped, just like a transition rule in `x-kubernetes-validations`.

- Allowing a list to only grow (each existing item must be preserved on update):

```
- expression: "oldSelf.items.all(x, x in self.items)"
  message: "items can only be appended to, not removed"
```

Scalar and array value validation

Validation of scalar values and arrays has the following features:

- If the property is a scalar (e.g., number or string), then in the `CEL` expression the `self` variable will be

that value.
- If the property is an array, then `self` will be an array, and you can use methods like `.size()`, `.all()`, `.exists()`, etc.

Example for an array:

```
type: object
properties:
  items:
    type: array
    items:
      type: string
    x-deckhouse-validations:
      - expression: "self.size() > 0"
        message: "The items list must not be empty"
```

Validation of additionalProperties (map)

For objects with additionalProperties (map), you can validate keys and values using methods like `.all(key, ...)`, `.exists(key, ...)`, etc.

Example:

```
type: object
properties:
  mymap:
    type: object
    additionalProperties:
      type: integer
    x-deckhouse-validations:
      - expression: "self.all(key, self[key] > 0)"
        message: "All values in mymap must be greater than 0"
```

values.yaml

This file is required for validating the source data when rendering templates without using extra Helm chart functions. Its closest analogs are Helm's schema files.

You can automatically add parameter validation from `config-values.yaml` to `values.yaml`. In this case, the basic `values.yaml` looks as follows:

```
x-extend:
  schema: config-values.yaml
type: object
properties:
  internal:
    type: object
    default: {}
```

templates

The `/templates` directory contains Helm templates.

- Use the path `.Values.<moduleName>` to access module settings in templates, and `.Values.global` for global settings. The module name is converted to `camelCase` notation.
- To facilitate working with templates, use lib-helm, which is a set of extra functions that make it easier to work with global and module values.
- Accesses to the registry from the ModuleSource resource are available at the `.Values.<moduleName>.registry.dockercfg` path.
- To use these functions to pull image pools in controllers, create a secret and add it to the corresponding parameter: `"imagePullSecrets": [{"name":"registry-creds"}]`.

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-creds
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: {{ .Values.<moduleName>.registry.dockercfg }}
```

A module can have parameters with which it can alter its behavior. Module parameters and their validation scheme are described in OpenAPI-schemes in `/openapi` directory.

The settings are stored in two files: `config-values.yaml` and `values.yaml`.

You can find an example of an OpenAPI schema in module template.

.helmignore

`.helmignore` allows you to exclude files from the Helm release. In case of DP modules, directories `/crds`, `/images`, `/hooks`, `/openapi` must be added to `.helmignore` to avoid exceeding 1 Mb limit of Helm release size.

Chart.yaml

This is a file for a chart, similar to `Chart.yaml` in Helm. It must contain at least a `name` parameter with the module name and a `version` parameter with the version. You don’t need to create this file, Deckhouse will create it automatically.

An example:

```
name: echoserver
version: 0.0.1
dependencies:
- name: deckhouse_lib_helm
  version: 1.38.0
  repository: https://deckhouse.github.io/lib-helm
```

module.yaml

The `module.yaml` file in the root of the module folder contains the module’s metadata.

The file might not be present, but it is recommended to fill it in. Most of the metadata will be available in the Module object in the cluster. The Module object will be created automatically after the module source (resource ModuleSource) is configured and synchronization is successful.

Parameters that can be used in `module.yaml`:

Parameter	Type	Description
`namespace`	String	Namespace where the module components will be deployed
`subsystems`	Array of strings	List of subsystems the module belongs to
`accessibility`	Object	Module accessibility settings
`accessibility.editions`	Object	Module operation settings in DP editions
`accessibility.editions.available`	Boolean	Defines whether the module is available in a DP edition
`accessibility.editions.enabledInBundles`	Array of strings	List of module bundles in which the module should be enabled by default
`descriptions`	Object	Arbitrary text description of the module’s purpose
`descriptions.en`	String	Description in English
`descriptions.ru`	String	Description in Russian

Parameter	Type	Description
`disable`	Object	Parameters related to the behavior when disabling a module
`disable.confirmation`	Boolean	Requires confirmation when disabling the module
`disable.message`	String	Message explaining what will happen when the module is disabled

If confirmation is required to disable a module (`confirmation` is set to `true`), then disabling the module is only possible if the corresponding ModuleConfig object has the annotation `modules.deckhouse.io/allow-disabling=true`. If this annotation is not present, Deckhouse will block the disabling attempt and display the warning message from the `message` field.

Other `module.yaml` parameters:

Parameter	Type	Description
`name`	String, mandatory parameter	Name of the module in Kebab Case. For example, `echo-server`
`exclusiveGroup`	String	If several modules share the same value of this parameter, only one of them can be active in the system at any given time. This prevents conflicts between modules performing similar or incompatible functions
`requirements`	Object	Module dependencies — a set of conditions that must be met for Deckhouse Platform (DP) to run the module
`requirements.deckhouse`	String	Dependency on the Deckhouse Platform version that the module is compatible with
`requirements.kubernetes`	String	Dependency on the Kubernetes version that the module is compatible with
`requirements.modules`	Object	Dependency on the version of other modules
`stage`	String	Module lifecycle stage. Possible values: `Experimental`, `Preview`, `General Availability`, `Deprecated`. If `stage` is set to `Experimental`, the module cannot be enabled by default. To allow the use of such modules, set the corresponding parameter to `true`
`tags`	Array of strings	Additional module tags. Tags are converted to labels of the Module object using the template `module.deckhouse.io/<TAG>=""`. For example, if you specify two tags `test` and `example`, the Module object will get the labels `module.deckhouse.io/test=""` and `module.deckhouse.io/example=""`
`weight`	Number	Module weight. Affects the module startup order: modules with a lower `weight` start earlier. Default: `900`. The startup order is also affected by module dependencies

Parameter	Type	Description
`critical`	Boolean	Marks the module as critical for the initial cluster bootstrap. Such modules (if used in the cluster) start during the initial cluster boot, before the cluster is considered fully ready. Other modules (non-critical) start only after the cluster is fully ready. Default: `false`

Example of metadata description for the `hello-world` module:

```
name: hello-world
tags: ["test", "myTag"]
weight: 960
stage: "Experimental"
namespace: "test"
exclusiveGroup: "group"
subsystems:
  - test
  - test1
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
descriptions:
  en: "The module to say hello to the world."
  ru: "Модуль, который приветствует мир."
requirements:
  deckhouse: ">= 1.61"
  kubernetes: ">= 1.27"
disable:
  confirmation: true
  message: "Disabling this module will delete all resources created by the module."
```

Configuring module accessibility in DP editions

The `accessibility` parameter lets you define the DP editions and module bundles in which a module is available, and whether it should be enabled by default.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Default
        - Managed
    ce:
      available: false
    ee:
      available: true
      enabledInBundles:
        - Managed
```

Parameter description:

- `accessibility`: *Object.* Root block for configuring module accessibility.
- `editions`: *Object.* Set of keys representing edition names. Each edition can have its own accessibility settings.
- `_default`: *Object.* Default configuration used when no settings are defined for a specific edition.
- `available`: *Boolean.* Defines whether the module is available in the specified edition.
- `enabledInBundles`: *Array of strings.* Module bundles in which the module is enabled by default. Supported module bundles (the full contents of each bundle are listed on this page):

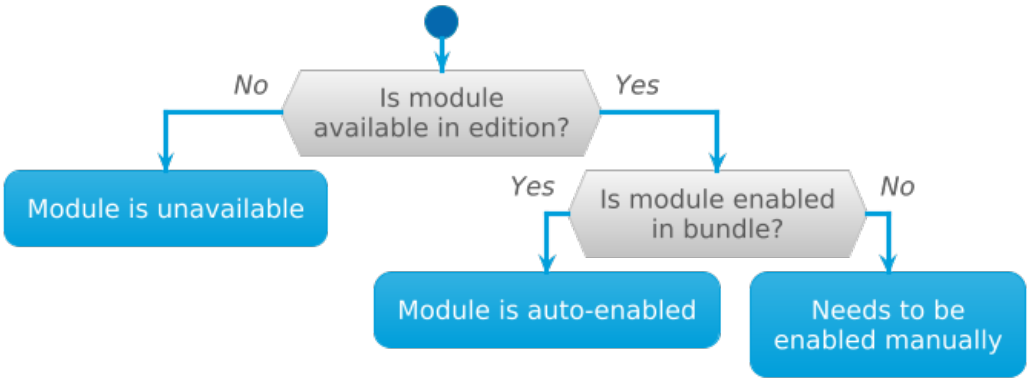
  - `Default`: Recommended set of modules for running a cluster. Includes monitoring, authorization control, networking, and other essential components.
  - `Managed`: Set of modules for clusters managed by cloud providers (for example, Google Kubernetes Engine).
  - `Minimal`: Minimal set that includes only the current module.

    > ⚠ Note that basic modules (such as the CNI module) are not included in this set. Without basic modules, DP can only operate in an already deployed cluster. The list of modules that must be enabled manually during installation is given in “Things to keep in mind when working with the Minimal module set”.

- Sections containing edition names. Define module behavior in specified editions. Possible values: `be`, `ce`, `ee`, `se`, `se-plus`.

Module accessibility logic

The following diagram illustrates the logic for determining module accessibility and whether it is enabled by default:



Configuration examples

In the following example, the module will be unavailable in all editions except DP Enterprise Edition. In DP Enterprise Edition, the module will be enabled by default in the `Managed` bundle.

```
accessibility:
  editions:
    _default:
      available: false
    ee:
      available: true
      enabledInBundles:
        - Managed
```

In the next example, the module will be available in all DP editions. In the `Managed` and `Default` bundles, the module will be enabled by default.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Managed
        - Default
```

In the following example, the module will be available in all DP editions. It will be enabled in the `Default` and `Managed` bundles for all editions, except for DP Basic Edition and DP Community Edition.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Default
        - Managed
    be:
      available: false
    ce:
      available: false
```

Additional resources

- Global custom resources
- Global parameters
- Module `deckhouse` parameters
- Module `multitenancy-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Build and publish

Deckhouse Platform (DP) uses container registry to pull a module and update it. The container registry stores module artifacts. Module artifacts are created when a module is built and can then be uploaded (published) to the registry.

Types of module artifacts

Module builds create three types of artifacts that are pushed to the container registry:

- **Images of application containers**. The build rules and source code for these images are stored in the images subdirectory of the *application name* directory. The built images are then specified in the templates and run in the cluster. Images are tagged with the content-based tags. Note that the lib-helm library must be enabled to use them in the templates.
- **Module image**. The module assembly rules are in the `werf.yaml` file in the module directory. The semantic versioning is used as image tags.
- **Release**. Module version artifact. Based on the release data, DP decides whether to update a module in the cluster. Releases have two types of tags: a semantic versioning tag (just like with the module image) and a tag that matches the release channel (e.g., `alpha`, `beta`, etc.). In the module template there is an example of workfklow for GitHub Actions, where the release is created automatically during building.

Building module artifacts and publishing them to the container registry

We suggest using the pre-compiled GitHub Actions to build module artifacts and upload them to the container registry as part of the CI/CD process.

The module template repository provides an example module. It contains a basic workflow for GitHub Actions and uses GitHub Packages (ghcr.io) as the container registry. The example workflow follows the logic outlined below:

- Build module artifacts when changes are made to the PR and when changes are merged into the main branch.
- Build module artifacts from tags using the production container registry.
- Publish module to GitHub Packages container registry to the selected release channel from a tag.

Module artifacts will be uploaded to `ghcr.io/<OWNER>/modules/`, which will serve as module source.

For the module workflow to run smoothly, adjust the properties of your project on GitHub as follows:

- Open the *Settings -> Actions -> General* page.
- Enable the *Read and write permissions* parameter in the *Workflow permissions* section.

You can also modify the workflow, use your own container registry, and design a more complex build and publish process (e.g., use separate container registries for development and production).

⚠ When developing **multiple modules** and publishing them to GitHub Packages, you should use your account's Personal Access Token (PAT).

Do not use `GITHUB_TOKEN` in GitHub Workflows. This way, you can avoid access rights issues when uploading images, as the final release images are stored at `ghcr.io/<OWNER>/modules/` — the address that belongs to the first repository created.

Here is how you can adapt a module template to use a PAT:

- On the *Settings -> Secrets and variables -> Actions* page create a Secret named `TOKEN` containing a PAT.
- Replace the `GITHUB_TOKEN` variable with `TOKEN` in `.github/workflows/`:

```
cd <REPO>
sed -i -e 's/GITHUB_TOKEN/TOKEN/g' $(find .github/workflows/ -type f)
```

i You can build module artifacts locally using werf (this may come in handy, for example, when debugging).

You can also set up a custom module artifact build and publish processes for your CI/CD system similar to the workflow for GitHub Actions provided in the module template. However, this may require an in-depth understanding of the module build and publish processes. Please contact the community should you have any questions or issues.

Here is a general scenario for using the workflow provided in the module template:

1. Publish the changes to the module code in the project branch on GitHub. This will trigger the module artifacts to be built and published to the container registry.
2. Create a new module release or add a tag in the semantic versioning format to the target commit.
3. Go to the *Actions* section of the module repository on GitHub and select *Deploy* in the workflow list on the left.
4. On the right side of the page, click on the *Run workflow* drop-down list, select the desired release channel, and enter the target tag in the tag input box. Click the *Run workflow* button.
5. Once the workflow is successfully executed, a new version of the module will be added to the container registry. The published version of the module can then be used in the cluster.

Additional resources

- Global custom resources

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

How to run and verify a module in the DP cluster

This section describes the process of running a module in a Deckhouse Platform (DP) cluster, as well as connecting Deckhouse Module Tools for setting up validation and metrics collection.

Run the module in the DP cluster

Follow these steps to run the module in a cluster:

- Define ModuleSource (the ModuleSource resource).
- *(optional)* Define the module update policy (the ModuleUpdatePolicy resource).
- Enable the module in the cluster (the ModuleConfig resource).

Module source

Deckhouse Platform (DP) can work with the following types of modules:

- Built-in modules. These are included in DP. Their release cycle is tied to the DP release cycle.

- Modules from module source. The release cycle for these modules is not tied to the DP release cycle.

Create a ModuleSource resource to set the source to fetch module information from. This resource will contain the address of the container registry to pull modules from, authentication parameters, and other settings.

An example of a ModuleSource resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.com/deckhouse/modules
    dockerCfg: <base64 encoded credentials>
```

After the ModuleSource resource is created, DP will start to perform periodic data synchronization with the module source (fetching information about the modules available in the source). By default, synchronization runs every 30 minutes. To change the interval, use the `scanInterval` parameter.

Use the following command to check the synchronization status:

```
d8 k get ms
```

If the synchronization is successful, you will see output similar to the one below:

```
$ d8 k get ms
NAME        COUNT   SYNC   MSG
example     2       16s    Ready
```

If there are synchronization errors, the `MSG` column will contain a general description of the error, e.g.:

```
$ d8 k get ms
NAME        COUNT   SYNC   MSG
example     2       16s    Some errors occurred. Inspect status for details
```

Detailed error information can be found in the `pullError` field in the status of the ModuleSource resource.

For example, here's how you can get detailed error description from the `example` module source:

```
$ d8 k get ms example -o jsonpath='{range .status.modules[*]}{.name}{" module error:\n\t"}
{.pullError}{"\n"}{end}'
module-1 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-
1/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
module-2 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-
2/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
```

If synchronization is successful, the `.status.modules` field of the ModuleSource resource will contain a list of modules ready to be enabled in the cluster.

Here is an example of how you can get a list of modules available from the `example` module source:

```
$ d8 k get ms example -o jsonpath='{.status.modules[*].name}'
module-1 module-2
```

The complete list of modules available from all module sources created in the cluster can be retrieved using the following command:

```
d8 k get ms  -o jsonpath='{.items[*].status.modules[*].name}'
```

After creating the ModuleSource resource and successful synchronization, *modules* — Module resources should start appearing in the cluster (DP creates them automatically, you do not need to create them). You can view the list of modules using the following command:

```
d8 k get module
```

Example of getting a list of modules:

```
$ d8 k get module
NAME       STAGE    SOURCE   PHASE       ENABLED   READY
module-one                   Available   False     False
module-two                   Available   False     False
```

To get additional information about the module, use the following command:

```
d8 k get module module-one -oyaml
```

Example of output:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-12-12T10:49:40Z"
  generation: 1
  name: module-one
  resourceVersion: "241504954"
  uid: 3ae75474-8e96-4105-a939-6df71cba82d8
properties:
  availableSources:
  - example
status:
  conditions:
  - lastProbeTime: "2024-12-12T10:49:41Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    message: disabled
    reason: Disabled
    status: "False"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-12T10:49:41Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    status: "False"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-16T15:46:26Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    message: not installed
    reason: NotInstalled
    status: "False"
    type: IsReady
  phase: Available
```

You can find available sources from which the module can be downloaded in the Module (there is only one in the example).

Next, you need to enable the module. To do this, you need to create a ModuleConfig with the name of the module.

The parameter `enabled` in ModuleConfig is responsible for enabling the module. If the module is available from multiple sources (resource ModuleSource), the required source can be specified in the `source` parameter.

The update policy (the name of the ModuleUpdatePolicy) can be specified in the `updatePolicy` parameter. It is not necessary to specify the update policy; in this case, it will be inherited from the Deckhouse Platform update parameters (releaseChannel and update of the ModuleConfig `deckhouse`).

Example of ModuleConfig for enabling the module `module-one` from the source `example`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-one
spec:
  enabled: true
  source: example
```

⚠ If there are mandatory parameters in the module configuration and the module is enabled without specifying them, a configuration validation error will occur. In this case, the `D8DeckhouseModuleValidationError` alert will appear, and the module will not be successfully activated.

To get more details, use the following command:

```
d8 k get mr -l module=<MODULE_NAME>
```

Make sure to specify the required configuration parameters in `ModuleConfig` according to the module's documentation.

After turning on the module, it should enter the download phase:

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE         ENABLED   READY
module-one           example  Downloading   False     False
```

⚠ If the module has not entered the download phase, check the module source (ModuleSource), as the module may not be able to download.

After a successful download, the module will enter the installation phase (`Installing`):

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE         ENABLED   READY
module-one           example  Installing    False     False
```

If the module was successfully installed, it will enter the `Ready` phase:

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE  ENABLED  READY
module-one           example  Ready  True     True
```

Example of a Module object in the cluster when the module has been successfully installed:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 1
  name: module-one
  resourceVersion: "242153004"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
  - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernetes: '> v1.25.0'
  source: example
  version: v0.7.24
  weight: 910
status:
  conditions:
  - lastProbeTime: "2024-12-12T15:49:35Z"
    lastTransitionTime: "2024-12-12T15:49:35Z"
    status: "True"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-17T09:35:27Z"
    lastTransitionTime: "2024-12-12T15:49:39Z"
    status: "True"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-17T09:35:27Z"
    lastTransitionTime: "2024-12-17T09:35:25Z"
    status: "True"
    type: IsReady
  - lastProbeTime: "2024-12-17T09:32:50Z"
    lastTransitionTime: "2024-12-17T09:32:50Z"
    status: "False"
    type: IsOverridden
  hooksState: 'v0.7.24/hooks/moduleVersion.py: ok'
  phase: Ready
```

In the Module, you can see the current installed version of the module, its size, the source from which it was downloaded, its dependencies, and the release channel.

In case of any errors, the module will enter the `Error` phase:

```
$ d8 k get module module-one
NAME         STAGE    SOURCE   PHASE  ENABLED  READY
module-one            example  Error  True     Error
```

If the enabled module has several available sources, and a source for the module is not explicitly selected in its ModuleConfig, the module will enter the `Conflict` phase:

```
$ d8 k get module module-one
NAME         STAGE    SOURCE   PHASE     ENABLED  READY
module-one                     Conflict  False    False
```

To resolve the conflict, specify the source of the module (ModuleSource name) explicitly in ModuleConfig.

After downloading the module, the module releases will appear in the cluster — ModuleRelease objects.

You can view the list of releases using the following command:

```
d8 k get mr
```

An example of retrieving the list of module releases:

```
$ d8 k get mr
NAME                         PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
module-one-v0.7.23           Superseded   deckhouse       33h
module-one-v0.7.24           Deployed     deckhouse       33h
module-two-v1.2.0            Superseded   deckhouse       48d
module-two-v1.2.1            Superseded   deckhouse       48d
module-two-v1.2.3            Deployed     deckhouse       48d
module-two-v1.2.4            Superseded   deckhouse       44d
module-two-v1.2.5            Pending      deckhouse       44d              Waiting for the
'release.deckhouse.io/approved: \"true\"' annotation
```

If the module release is in the `Superseded` status, it means that the module release is outdated, and there is a newer release that has replaced it.

⚠ If a module release is `Pending`, it means that manual confirmation is required to install it (see module update policy below). You can confirm the module release using the following command (specify the moduleRelease name):

```
d8 k annotate mr <module_release_name> modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve <module-name> <version>
```

Switching the module to a different module source

Follow these steps to deploy a module from a different module source:

1. Create a new ModuleSource resource.
2. Specify it in the `source` field in ModuleConfig.
3. Ensure that new module releases (ModuleRelease objects) are created from the new module source in accordance with the update policy:

   ```
   d8 k get mr
   ```

Module update policy

The module update policy refers to the rules that DP uses to update modules in the cluster. It is set by the ModuleUpdatePolicy resource with the following settings:

- module update mode (automatic, manual, updates are disabled);
- the release channel to use for updates;
- time windows for automatic updates during which the module update is permitted.

You do not have to create the ModuleUpdatePolicy resource. If the update policy for a module is not defined (there is no corresponding ModuleUpdatePolicy resource), the update settings match the update settings of DP (the update parameter of the `deckhouse` module).

Example of the ModuleUpdatePolicy resource, whose update policy allows automatic module updates on Mondays and Wednesdays from 13:30 to 14:00 UTC:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - "Mon"
      - "Wed"
      from: "13:30"
      to: "14:00"
```

The update policy is specified in the `updatePolicy` field in ModuleConfig.

Enabling the module

Before enabling the module, make sure that it can be enabled. Run the following command to list all the available DP modules:

```
d8 k get modules
```

The module must be in the list.

Below is an example of the output:

```
$ d8 k get module
NAME        STAGE    SOURCE   PHASE       ENABLED   READY
...
module-one                    Available   False     False
module-two                    Available   False     False
...
```

It shows that the `module-one` module can be enabled.

If the module is not in the list, check that module source is defined and the module is listed in the module source. Also check the update policy of the module (if defined). If the module update policy is not defined, it matches the DP update policy (the releaseChannel parameter and the update section of the `deckhouse` module parameters).

You can enable the module similarly to built-in DP modules using any of the following methods:

- Run the command below (specify the name of the module):

  ```
  d8 system module enable <MODULE_NAME>
  ```

- Create a `ModuleConfig` resource containing the `enabled: true` parameter and module settings..

  Below is an example of a ModuleConfig that enables and configures the `module-one` module in the cluster:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: module-one
  spec:
    enabled: true
    settings:
      parameter: value
    version: 1
  ```

Troubleshooting

If there were errors while enabling a module in the cluster, you can learn about them as follows:

- View Deckhouse log:

```
d8 k -n d8-system logs -l app=deckhouse
```

- View the Module object in more detail:

```
d8 k get module module-one -oyaml
```

- View the ModuleConfig object of the module.

  Here is an example of the error message for `module-one`:

```
$ d8 k get moduleconfig module-1
NAME         ENABLED   VERSION   AGE   MESSAGE
module-one   true                7s    Ignored: unknown module name
```

- View the ModuleSource object.

  Example output if the module source has problems with downloading the module:

```
$ d8 k get ms
NAME         COUNT   SYNC   MSG
example      2       16s    Some errors occurred. Inspect status for details
```

Similar to DeckhouseRelease (a DP release resource), modules have a ModuleRelease resource. DP creates ModuleRelease resources based on what is stored in the container registry. When troubleshooting module issues, check the ModuleRelease available in the cluster as well:

```
d8 k get mr
```

Output example:

```
$ d8 k get mr
NAME                  PHASE        UPDATE POLICY          TRANSITIONTIME   MESSAGE
module-1-v1.23.2      Pending      example-update-policy  3m               Waiting for
the 'release.deckhouse.io/approved: "true"' annotation
```

The example output above illustrates ModuleRelease message when the update mode (update.mode of the ModuleUpdatePolicy is set to `Manual`. In this case, you must manually confirm the installation of the new module version by adding the `modules.deckhouse.io/approved="true"` annotation to the ModuleRelease:

```
d8 k annotate mr module-1-v1.23.2 modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve module-1 v1.23.2
```

Integrating Deckhouse Module Tools for Module Validation

To enable automatic validation of the module structure and, if needed, metrics reporting, you can integrate Deckhouse Module Tools (DMT) into your build process.

For GitHub Projects

A dedicated GitHub Action is available for integrating the DMT into your module.

To connect the DMT, add the following step to your build workflow configuration in `[project].github/workflows/build.yml`:

```
jobs:
  lint:
    runs-on: ubuntu-latest
    continue-on-error: true
    name: Linting
    steps:
      - uses: actions/checkout@v4
      - uses: deckhouse/modules-actions/lint@main
      env:
         DMT_METRICS_URL: ${{ secrets.DMT_METRICS_URL }}
         DMT_METRICS_TOKEN: ${{ secrets.DMT_METRICS_TOKEN }}
```

The `DMT_METRICS_URL` and `DMT_METRICS_TOKEN` variables are optional. If set, the DMT will send telemetry to the specified endpoint.

> **i** If the module resides in the `deckhouse` GitHub organization, these variables will be automatically populated from the configured secrets.

A complete example of the configuration can be found in the build_dev.yml file.

To simplify your setup, you can also use the provided configuration templates.

For GitLab Projects

For GitLab projects, ready-to-use templates are available and can be included in your `.gitlab-ci.yml` file to automatically configure the build and validation processes:

- **Setup**: Setup configuration template
- **Build**: Build process configuration template

Steps to connect

1. In your project's `.gitlab-ci.yml` file, add references to the templates:

   ```
   include:
     - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-
   ci/refs/heads/main/templates/Setup.gitlab-ci.yml
     - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-
   ci/refs/heads/main/templates/Build.gitlab-ci.yml
   ```

2. After adding the templates, in the same `.gitlab-ci.yml` configuration, add a step to perform the check:

   ```
   Lint:
     extends: .lint
   ```

Additional resources

- Global custom resources
- Module `deckhouse` parameters

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Dependencies of the Deckhouse Platform modules

This section covers the dependencies you can set for the module.

Dependencies are a set of conditions (requirements) that must be met in order for Deckhouse Platform to be able to run a module.

Deckhouse Platform (DP) supports the following module dependencies:

- Deckhouse Platform version;
- Kubernetes version;
- version of other modules.

Deckhouse Platform version dependency

This dependency defines the minimum or maximum DP version with which the module is compatible.

An example of setting up a dependency for Kubernetes 1.27 and higher in the `module.yaml` file:

```
name: test
weight: 901
requirements:
   deckhouse: ">= 1.61"
```

> **i** For testing, you can set the `TEST_EXTENDER_DECKHOUSE_VERSION` environment variable to imitate the desired version of Deckhouse Platform.

Deckhouse checks whether the dependency is met in the following cases:

1. **When installing or upgrading a module** If the DP version does not meet the requirements specified in the release module dependencies, the latter will not be installed or upgraded.

   Below is an example of the ModuleRelease resource for which the DP version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get mr
   ```

   Output information:

   ```
      NAME                    PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
      test-v0.8.3             Pending      test-alpha      2m30s            requirements are not satisfied:
   current deckhouse version is not suitable: 1.0.0 is less than or equal to v1.64.0
   ```

2. **When upgrading Deckhouse Platform** Deckhouse checks if the new DP version matches the dependencies of the installed and active modules. If at least one module is not compatible with the new version, the DP upgrade will not be performed.

   Below is an example of the DeckhouseRelease resource for which the DP version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
   ```

   Output information:

   ```
   NAME                    PHASE         TRANSITIONTIME   MESSAGE
   v1.73.3                 Skipped       74m
   v1.73.4                 Pending       2m13s            requirements of test are not satisfied: v1.73.4
   deckhouse version is not suitable: v1.73.4 is greater than or equal to v1.73.4
   ```

3. **When conducting initial module analyses** Deckhouse checks the current version of DP and the dependencies of the installed modules. If a mismatch is discovered, the module will be disabled.

Kubernetes version dependency

This dependency defines the minimum or maximum Kubernetes version with which the module is compatible.

Here is how you can enable the Kubernetes version dependency for Kubernetes 1.28 and higher in the `module.yaml` file:

```
name: test
weight: 901
requirements:
   kubernetes: ">= 1.28"
```

> **i** For testing, you can set the `TEST_EXTENDER_KUBERNETES_VERSION` environment variable to imitate the desired version of Deckhouse Platform.

Deckhouse checks whether the dependency is met in the following cases:

1. **When installing or upgrading a module** If the Kubernetes version does not meet the requirements specified in the release module dependencies, the latter will not be installed or upgraded.

   Below is an example of the ModuleRelease resource for which the Kubernetes version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get modulereleases.deckhouse.io
   ```

   Output information:

   ```
   NAME                          PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
   test-v0.8.2                   Pending      test-alpha      24m              requirements are not
   satisfied: current kubernetes version is not suitable: 1.29.6 is less than or equal to 1.29
   virtualization-v.0.0.0-dev4   Deployed      deckhouse      142d
   ```

2. **When upgrading Kubernetes** Deckhouse examines the dependencies of active modules, and if at least one module is incompatible with the new Kubernetes version, the version upgrade will not proceed.

   Below is an example of the output you may encounter when a module is incompatible with a newer version of Kubernetes:

   ```
   root@dev-master-0:~# d8 system edit cluster-configuration
   ```

   Output information:

   ```
   Save cluster-configuration back to the Kubernetes cluster
   Update cluster-configuration secret
   Attempt 1 of 5 |
           Update cluster-configuration secret failed, next attempt will be in 5s"
           Error: admission webhook "kubernetes-version.deckhouse-webhook.deckhouse.io" denied the request:
   requirements of test are not satisfied: 1.27 kubernetes version is not suitable: 1.27.0 is less than or
   equal to 1.28
   ```

3. **When conducting initial module analyses** If the Kubernetes version does not conform to the dependencies of the modules that are already installed, DP will disable those modules.

4. **When upgrading Deckhouse Platform** Deckhouse checks the default Kubernetes version value for DP and if it is not compatible with the active modules, the DP update will not be carried out.

   Below is an example of the DeckhouseRelease resource for which the Kubernetes version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
   ```

   Output information:

   ```
   NAME                    PHASE         TRANSITIONTIME   MESSAGE
   v1.73.3                 Pending       7s              requirements of test are not satisfied: 1.27
   kubernetes version is not suitable: 1.27.0 is less than or equal to 1.28
   ```

Dependency on the version of other modules

Dependencies on other modules describe the conditions for enabling, updating, and disabling a module. A module in the Deckhouse Platform may have required and conditional dependencies on versions of other modules.

Required dependencies

This dependency defines the list of **enabled** modules and their versions that are required for the module to work.

> **i** The built-in DP module version is considered equal to the DP version.

If you need to specify that some module is simply enabled, no matter what version, then you can use the following syntax (using the `user-authn` module as an example):

```
requirements:
  modules:
    user-authn: ">= 0.0.0"
```

Example of setting up a dependency on three modules:

```
name: hello-world
requirements:
  modules:
    ingress-nginx: '> 1.67.0'
    node-local-dns: '>= 0.0.0'
    operator-trivy: '> v1.64.0'
```

Conditional module requirements

✖ Conditional dependencies for modules are available for Deckhouse Platform starting with version 1.73. If you need to use them for a module, set a Deckhouse Platform version dependency 1.73 or higher for that module.

Use this when your module works alone, but integrates with another module **if it is present**.

i Conditional dependencies may affect the ability to enable, disable, and update both modules: the dependent module and the module on which it depends.

To mark a requirement as conditional, add `!optional` to the version constraint string:

```
name: prometheus
requirements:
  modules:
    test: ">v0.22.1 !optional"
```

i The following sections describes restrictions on the use of conditional module dependencies and provides examples of settings where:

- `prometheus` — the target module for which an conditional dependency is specified;
- `test` — a module that can be used in conjunction with the target module.

Restrictions on enabling and disabling prometheus when there is an conditional dependency on test

When there is an conditional dependency on the version of another module, the target module has the following restrictions on enabling and disabling:

1. If `test` is not enabled in the cluster, `prometheus` can be enabled.

   Example: `test` is disabled + `prometheus` has an conditional requirement `test: ">v0.22.1 !optional"` → `prometheus` will be enabled, the requirement is skipped.

2. If `test` is already enabled in the cluster, enabling `prometheus` with `requirements` is only possible if `requirements` specifies requirements that the current version of `test` meets.

   Example: `test` version `v0.21.1` is enabled in the cluster + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → installing/enabling `prometheus` will result in a dependency mismatch error (the current version of `test` does not match `requirements`).

3. If `test` is disabled in the cluster, `prometheus` will remain enabled.

   Example: `test` is enabled in the cluster + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → disabling `test` will be successful, `prometheus` will not be disabled.

Restrictions on updating prometheus when there is an conditional dependency on test

When there is an conditional dependency on the version of another module, the target module has the following update restrictions:

1. `prometheus` can be updated even if the cluster does not have the `test` module.

   Example: `test` is disabled + `prometheus` has an conditional requirement `test: ">v0.22.1 !optional"` → `prometheus` will be updated.

2. The update for `prometheus` will be blocked until `test` specified in `requirements` is updated in the cluster.

   Example: `test` is enabled + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → `prometheus` will not be updated until `test` is updated.

Restrictions on enabling test, whose version is specified in prometheus dependencies

If the `prometheus` module is enabled, it is not possible to enable `test`, whose version does not match the expression specified in `requirements` for `prometheus`.

Example: `prometheus` is included + an conditional requirement `test: ">v0.22.1 !optional"` is specified for `prometheus` → an attempt to include `test v0.21.1` will result in a dependency mismatch error (the version `test v0.21.1` does not match the condition in `requirements` for `prometheus`).

Restrictions on updating test, whose version is specified in prometheus dependencies

If prometheus and test are included in the cluster, test can only be updated to a version that meets the requirements specified in requirements for prometheus.

Example: The `prometheus` and `test` modules are included in the cluster + an conditional requirement `test: '=v0.22.1 !optional'` is set for `prometheus` + an attempt to update `test` to version `0.23.1` → `test` will not be updated because the required version does not meet the `requirements` for `prometheus`.

⚠

- Enabling or disabling modules may take longer because of extra extender checks.
- Known limitation: during reconciliation the list of enabled modules may briefly be empty, which in rare cases lets an conditional dependency check pass incorrectly. If this happens, retry the operation.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Versioning modules

Deckhouse Platform (DP) uses semantic versioning for modules.

When choosing a version, stick to the following guidelines:

- updating the **patch version** (e.g., from `0.0.1` to `0.0.2`) means that some issue has been fixed;
- updating the **minor version** (e.g., from `0.0.1` to `0.1.0`) means that some new feature has been added;
- updating the **major version** (e.g., from `0.0.1` to `1.0.0`) means that a feature has been added that radically changes the module's capabilities; the interface has undergone significant changes or a major phase of development has been completed.

The git tag and registry container **always** have a "v" before the version number, e.g., `v0.0.73` or `v1.0.0`.

Release channels

Release channels allow you to publish a module version not for all users at once, but for a limited group of users. You define the stability level of the module version and the maximum release channel to which it can be published.

Note that the choice of a specific release channel does not determine how stable the module is. The channels are an instrument for delivery of updates and determine the degree of stability of a particular release.

When publishing a new version of the module to the release channel, first use the `Alpha` release channel. Then, if module version has no issues, publish the module version sequentially to other release channels, taking their stability into account: `Alpha` → `Beta` → `Early Access` → `Stable` → `Rock Solid`. If the module

version requires fixing, the publication of that version should be halted. After releasing a version with fixes, the publication stage must be repeated starting from the `Alpha` release channel.

Procedure for publishing a new version

The recommended sequence of publication of the module version in release channels:

1. Publish a new version of the module in the `Alpha` release channel.
2. If the version works stably, publish it in the following release channels sequentially: `Beta` → `Earlyaccess` → `Stable` → `Rocksolid`.
3. If errors arise, stop the publication and correct them.
4. Repeat the publication of the version, starting from the release channel `alpha`.

Module lifecycle

During its life cycle, the module goes through the following stages:

- **Experimental** refers t to an experimental version. The module functionality may undergo significant changes. Compatibility with future versions is not guaranteed.

  > **i** Modules in the `Experimental` stage cannot be enabled by default. To allow the use of such modules, set the allowExperimentalModules parameter to true.

- **Preview** refers to a preliminary version. The module functionality may change, but the basic features will be preserved. Compatibility with future versions is ensured, but may require additional migration actions.
- **General Availability (GA)** refers to a generally available version. The module is ready to be used in production environments.
- **Deprecated** refers to a module version that has been deprecated. Further development and support of the module is discontinued.

Definition of module stability

Depending on the stage of the module lifecycle and the release channel from which the specific module version was installed, the overall stability can be determined by the following table:

Lifecycle Stage	Release Channels				
	Alpha	**Beta**	**Early Access**	**Stable**	**Rock Solid**
Experimental	Experiments	Experiments	Experiments	Beta testing	Beta testing
Preview	Experiments	Limited use	Limited use	Production use	Production use
General Availability	Experiments	Limited use	Limited use	Production use	Production use in critical systems
Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated

- **Experiments** — Functionality checks, experiments, and testing;
- **Beta testing** — Functionality checks, experiments, and testing. Limited use by experienced users in environments equivalent to production;
- **Limited use** — Development environments, pilot projects, low-impact production environments;
- **Production use** — Production environments and those equivalent to them;
- **Production use in critical systems** — Mission-critical production environments and those equivalent to them;
- **Deprecated** — Should be phased out of use.

Highlights:

- A module in `Experimental` status on the `Stable` channel is recommended to be used in production environments only with limitations.
- A module in `General Availability` status on the `Alpha` channel is also not recommended for use in production environments.
- Only `GA` modules installed from `Early Access`, `Stable`, or `Rock Solid` channels are suitable for production environments.
- `Deprecated` modules are recommended to be replaced.

API versioning

Modules in DP use custom resources to interact with users. The `apiVersion` parameter with the API version of these resources is set according to the following rules:

- `v1alphaX` refers to an API that has just been published. This API needs to be tested to see how user-friendly it is, as well as how valid and consistent its settings are.
- `v1betaX` refers to the API that has passed initial testing. Its logical development and refinement is in progress.
- `v1stableX` refers to a stable API. At this point, its fields are kept in the specification while its validation rules do not change to be more stringent.

If necessary, a new version of API v2 can be released, which goes through the same stages but with the prefix `v2`. It's important to remember that after the release of the `v1stableX` version, Kubernetes will consider it more prioritized than the `alpha` or `beta` versions until a new stable version `v2stableX` is released. When executing `kubectl apply` and `kubectl edit` commands, the `v1stableX` version will be used by default.

Release of the new API version

The reasons for releasing a new version may be as follows:

- structure changes;
- updating obsolete parameters.

You can add new parameters without changing the version.

Automatic API Conversion

To enable automatic conversion of module parameters from one version to another, you can include the appropriate module conversions. Conversions may be necessary when a parameter is renamed or moved to a different location in a new version of the OpenAPI specification.

Recommendations for Releasing New Versions of CRD

Please follow these recommendations when releasing a new version of the CustomResourceDefinition (CRD):

- Set the parameter `deprecated: true` for previous versions of the CRD. For more details on working with deprecated versions of CRD, refer to the Kubernetes documentation.
- Do not change the storage version, in which data is stored inside etcd, until at least two months have passed since the release of the new version.

Additional resources

- Module `deckhouse` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

An example of creating a module with adaptation of an existing chart

This section provides an example of creating a `helloworld` module based on module template and an adaptation of the hello-world Helm-chart.

Preparing module source code and building

1. Install the tools you will need:
   - git
   - sed
   - yq
2. Fork the module template repository or copy it.

   ```
   git clone git@github.com:deckhouse/modules-template.git helloworld \
     && cd helloworld
   ```

3. Enter the name of the module in the `module.yaml` file.

   We will use the `helloworld` module name in this example, but you can use any name you like. For this, replace 'helloworld' in the commands and in your repo's name with the one you prefer.

> **i** Note that the module name may differ depending on the command. In may be written in *kebab-case* as well as *camelCase*. If you use a custom module name, make sure to modify it accordingly.

Run the following command to add the module name to the `module.yaml` file or edit it manually:

```
sed -i -e 's/^name:.*$/name: helloworld/' module.yaml
```

4. Clone the hello-world chart source code into a temporary directory.

```
git clone https://github.com/giantswarm/hello-world-app .tmp-chart
```

5. Copy the chart templates to the `templates` directory of the module (make sure to empty it beforehand).

```
rm -rf templates/*
cp -fR .tmp-chart/helm/hello-world/templates/* templates/
cp .tmp-chart/helm/hello-world/values.yaml values.yaml
```

6. Replace the `.Values` path in the chart templates with `.Values.helloworld`.

> **i** This is due to the architectural feature of addon-operator. You have to stick to it to be able to access module values.

```
sed -i -e 's/.Values/.Values.helloworld/g' $(find templates/ -type f)
```

7. Add the OpenAPI schema of the module settings.

   The module parameters are specified in the OpenAPI schema in the openapi directory. Execute the following command to convert the JSON schema of the chart parameters to the OpenAPI schema of the module:

```
jq 'walk(
   if type == "object" and .type == "object" and (keys | length) == 1
   then . + {additionalProperties: true}
   else .
   end
)' .tmp-chart/helm/hello-world/values.schema.json > openapi/config-values.yaml
```

8. Define a rule for building an application container image.

   The rules for building application container images must reside in a subdirectory of the images directory of the module. Run the following commands to create an application image directory and a Dockerfile with the image build rules:

```
rm -rf images/*
mkdir images/helloworld
echo "FROM quay.io/giantswarm/helloworld:0.2.0" > images/helloworld/Dockerfile
```

9. Replace the image in the Deployment manifest with the Deckhouse Platform library helper. This will allow you to use the current content-based image tag.

```
sed -Ei 's/image\:(.*)/image: {{ include "helm_lib_module_image" (list .
"helloworld") }}/g' templates/deployment.yaml
```

10. Delete module hooks, CRDs, and temporary files.

    In our example, hooks and CustomResourceDefinitions are not used. Run the following commands to clear the `hooks` and `crds` directories:

```
rm -rf hooks/
rm -rf crds/
rm -rf .tmp-chart
```

11. Set up CI/CD.

    The project template in the `.github` directory contains pre-built GitHub Actions workflow files that implement a simple scheme for building and publishing a module using the GitHub Packages registry (ghcr.io). Module artifacts will be pushed to `ghcr.io/<OWNER>/modules/`, which will be the module source. Make changes to the workflow files if the suggested option does not work for you.

    Do the following in the properties of your project on GitHub to make the module workflow run smoothly:

    - Open the *Settings -> Actions -> General* page.
    - Enable *Read and write permissions* in the *Workflow permissions* section.

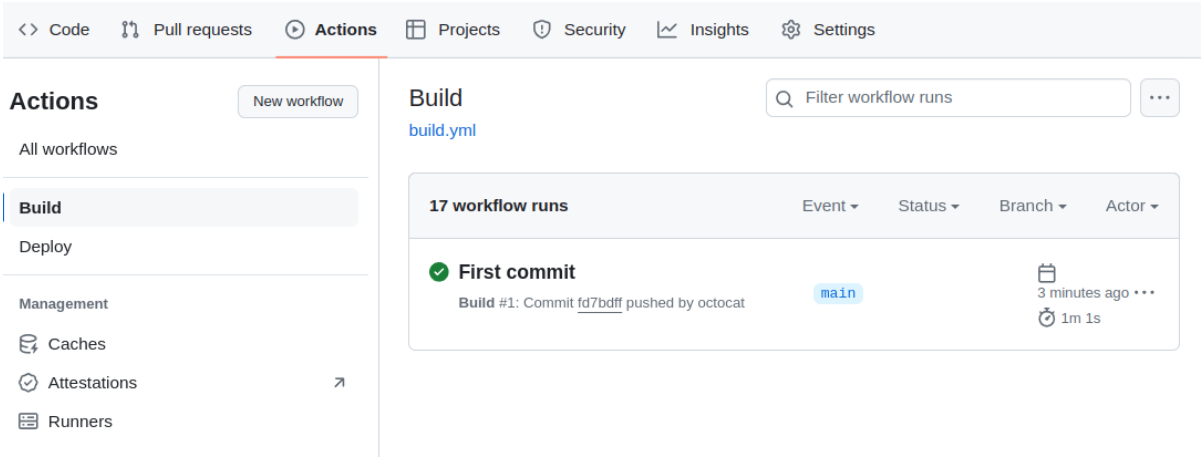
12. Commit the changes to the repository (specify the address of the module's Git repository).

    ```
    git add .
    git commit -m "Initial Commit"
    git push --set-upstream origin <GIT_REPO_URL>
    ```

13. Confirm that the module was built successfully.

    Go to the *Actions* section of the module repository and select *Build* from the workflow list on the left. The workflow that starts after you have executed the `git push` command at the previous step is expected to run successfully.

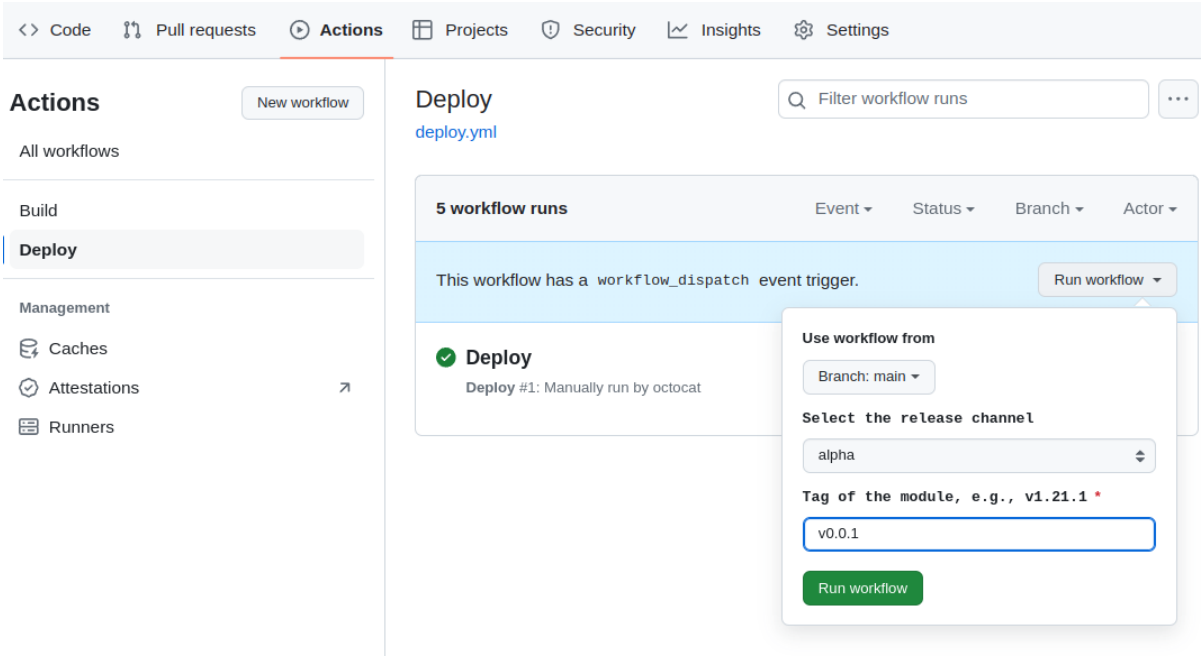
    An example:



Publishing the module to the release channel

Here's how you can publish version `v0.0.1` of the module to the *Alpha* release channel:

1. Create a new release `v0.0.1` of the module in the GitHub repository or add the `v0.0.1` tag.
2. Go to the *Actions* section of the module repository and select *Deploy* in the workflow list on the left.
3. Click on the *Run workflow* drop-down list on the right of the page and select `alpha`. Enter the `v0.0.1` tag in the tag input field. Click the *Run workflow* button.



4. Confirm that the module publishing workflow has completed successfully.

You can now access your module in a cluster managed by Deckhouse Platform.

Enabling a module in a cluster

The following is the sequence of steps to enable the `helloworld` in a cluster managed by Deckhouse Platform.

1. Create an access token in the GitHub repository with permissions to handle GitHub Packages.
2. Generate an authentication string to access the GitHub Packages container registry in dockerconfigjson format; insert your GitHub username (or organization) and access token in the angle brackets below:

```
base64 -w0 <<EOF
{
  "auths": {
    "ghcr.io": {
      "auth": "$(echo -n '<OWNER>:<TOKEN>' | base64 -w0)"
    }
  }
}
EOF
```

3. Create a ModuleSource resource in the cluster (specify the container registry address and authentication string).

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: ghcr
spec:
  registry:
    # Insert your GitHub username (or organization), e.g., ghcr.io/octocat/modules.
    repo: ghcr.io/<!OWNER>/modules
    # Paste the authentication string for accessing GitHub Packages that you got in
the previous step
    dockerCfg: <!REGISTRY_CREDENTIALS>
EOF
```

   Note that it may take a few seconds to synchronize data after a resource has been created.

4. Check the list of available modules

```
$ d8 k get module
NAME       STAGE    SOURCE   PHASE       ENABLED   READY
...
helloworld                   Available   False     False
...
```

5. Create a ModuleUpdatePolicy resource that defines the module update policy.

   Run the following command to create an update policy with the *Alpha* release channel and *Auto* update mode:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: helloworld-policy
   spec:
     releaseChannel: Alpha
     update:
       mode: Auto
   EOF
   ```

6. Create a ModuleConfig where you specify the module source (the `source` parameter), the update policy (the `updatePolicy` parameter), and set the `enabled` parameter to `true`:

   ```
   d8 k apply -f - <<EOF
      apiVersion: deckhouse.io/v1alpha1
      kind: ModuleConfig
      metadata:
        name: helloworld
      spec:
         enabled: true
         source: ghcr
         updatePolicy: helloworld-policy
   ```

7. Check the ModuleSource (the status should have no errors and list the available modules):

   ```
   d8 k get ms ghcr -o yaml
   ```

8. Confirm that the new ModuleRelease objects have been created for the module:

   ```
   d8 k get mr
   ```

   An example of output:

   ```
   $ d8 k get mr
   NAME                                PHASE        UPDATE POLICY
   TRANSITIONTIME   MESSAGE
   helloworld-v0.0.1                   Deployed     helloworld-policy    22m
   ```

9. If the release has been successfully installed, wait for the Deckhouse Platform pod to restart.

   ```
   d8 k -n d8-system get pod -l app=deckhouse
   ```

   After a while, the module objects will be available in the cluster.

   If you run into errors while starting the module, check the DP log:

   ```
   d8 k -n d8-system logs deploy/deckhouse -f | jq -rc '.msg'
   ```

or check the status of the DP queue:

```
d8 system queue list
```

Migration of ModuleUpdatePolicy to version v1alpha2

If there is a ModuleUpdatePolicy of version v1alpha1 in the cluster, the following migration steps to version v1alpha2 must be performed:

If any ModuleUpdatePolicy of version v1alpha1 in the cluster defines `moduleReleaseSelector`, alerts `ModuleHasDeprecatedUpdatePolicy` will be triggered in the monitoring system for all modules that match this selector. In this case, follow these steps to migrate to version v1alpha2 of ModuleUpdatePolicy:

- Specify the update policy for the corresponding modules in the ModuleConfig `spec.updatePolicy` parameter.
- Execute the following command, specifying the required ModuleUpdatePolicy:

```
d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io <MUP_NAME> --type='json' \
  -p='[{"op": "replace", "path":
"/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}}]'
```

Additional resources

- Global custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module development and debugging

When developing modules, you may want to pull and deploy a module bypassing the release channels. The ModulePullOverride resource is used for this purpose.

> ⚠ The ModulePullOverride resource is intended **for development and debugging environments only**. Using it in production clusters is *not recommended*. Support for the resource might be removed in future Deckhouse Platform versions.

An example of ModulePullOverride:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: <module-name>
spec:
  imageTag: <tag of the module image>
  scanInterval: <image digest check interval. Default: 15s>
```

The `maintenance mode` can also be used for development purposes. In this mode, Deckhouse disables resource management for the module and does not apply changes automatically. This mode is not intended for use in production clusters.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
...
spec:
  enabled: true
  maintenance: NoResourceReconciliation
  settings:
```

Requirements for the resource parameters:

- The module name (`metadata.name`) must match the module name in the ModuleSource (`.status.modules.[].name`).

- The container image tag (`spec.imageTag`) can be anything, e.g., `pr333`, `my-branch`.

The `spec.scanInterval` time interval (optional) defines the interval for scanning images in the registry. The default interval is 15 seconds. To force scan you can change the interval or set the `modules.deckhouse.io/renew=true` annotation on ModulePullOverride. The annotation forces the module to be re-pulled and redeployed even when the image digest has not changed, and is removed once the redeploy succeeds.

The `spec.rollback` indicates whether the deployed module release should be rollback after deleting the ModulePullOverride.

You can get the result of applying ModulePullOverride in the message (column `MESSAGE`) when retrieving ModulePullOverride information. The value `Ready` indicates the successful application of ModulePullOverride parameters. Any other value indicates conflict.

Example of absence of conflicts when using ModulePullOverride:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME      UPDATED   MESSAGE   ROLLBACK
example1  10s       Ready     false
```

Requirements for the module:

- The module must exist; otherwise the message for ModulePullOverride will be *The module not found*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME      UPDATED   MESSAGE                ROLLBACK
  example1  10s       The module not found   false
  ```

- The module must not be embedded Deckhouse module; otherwise the message in ModulePullOverride will be *The module is embedded*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME           UPDATED  MESSAGE                  ROLLBACK
  ingress-nginx  10s      The module is embedded   false
  ```

- The module must be enabled; otherwise, the message for ModulePullOverride will be *The module disabled*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME     UPDATED   MESSAGE               ROLLBACK
  example  7s        The module disabled   false
  ```

- The module must have a source; otherwise the message at ModulePullOverride will be *The module does not have an active source*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME       UPDATED   MESSAGE                                      ROLLBACK
  example    12s       The module does not have an active source   false
  ```

- The source for the module must exist; otherwise the message for ModulePullOverride will be *The source not found*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME       UPDATED   MESSAGE                 ROLLBACK
  example    12s       The source not found    false
  ```

To update the module without waiting for the next update cycle to begin, you can execute the following command:

```
d8 k annotate mpo <name> modules.deckhouse.io/renew=true --overwrite
```

Module availability and enabling by default

To assign Deckhouse editions the module should be available in, as well as module bundles where it should be enabled by default, use the `accessibility` field in `module.yaml`:

```
name: test
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
```

In this configuration, the module will be available in `ee` (DP Enterprise Edition)
and can be enabled using the ModuleConfig object, and will be enabled by default in the `Default` bundle.

How ModulePullOverride works

After creating ModulePullOverride, the corresponding module will not consider ModuleUpdatePolicy, and will also not load and create ModuleRelease objects. The module will be loaded upon every change of the `imageDigest` parameter, after which it will be applied in the cluster. The ModuleSource status will have `overridden: true`, which indicates that ModulePullOverride is being used instead of ModuleUpdatePolicy. Also, the corresponding Module object will have an `IsOverridden` field in its status, and the module version from `imageTag`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 16
  labels:
    deckhouse.io/epoch: "1326105356"
  name: example
  resourceVersion: "230347744"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
  - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernets: '> v1.30.0'
  source: example
  version: mpo-tag
  weight: 910
status:
  conditions:
  - lastProbeTime: "2024-12-03T15:57:20Z"
    lastTransitionTime: "2024-12-03T15:57:20Z"
    status: "True"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-03T15:59:58Z"
    lastTransitionTime: "2024-12-03T15:57:26Z"
    status: "True"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-03T15:59:58Z"
    lastTransitionTime: "2024-12-03T15:56:23Z"
    status: "True"
    type: IsReady
  - lastProbeTime: "2024-12-03T15:59:48Z"
    lastTransitionTime: "2024-12-03T15:56:47Z"
    status: "True"
    type: IsOverridden
  phase: Ready
```

The module will keep running after ModulePullOverride is removed. But if there is a ModuleUpdatePolicy for the module, new releases of the module (ModuleRelease) will be pulled to replace the current “developer version”.

An example

1. Suppose there are two modules, `echo` and `hello-world`, defined in ModuleSource. The update policy is set for them, and they are pulled in and installed in DP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
  - name: echo
    policy: test-alpha
  - name: hello-world
    policy: test-alpha
  modulesCount: 2
```

2. Enable the module and create ModulePullOverride for the `echo` module:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354
```

After creating ModulePullOverride, the image tag `registry.example.com/deckhouse/modules/echo:main-patch-03354` will be used for the module (`ms:spec.registry.repo/mpo:metadata.name:mpo:spec.imageTag`).

1. The ModulePullOverride will change with each update of the module:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354
  scanInterval: 15s
status:
  imageDigest:
sha256:ed958cc2156e3cc363f1932ca6ca2c7f8ae1b09ffc1ce1eb4f12478aed1befbc
  message: "Ready"
  updatedAt: "2023-12-07T08:41:21Z"
```

   where:

   - `imageDigest` is the unique identifier of the container image that was pulled.
   - `lastUpdated` is the time when the image was last pulled.

2. In this case, ModuleSource would look as follows:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
  - name: echo
    overridden: true
  - name: hello-world
    policy: test-alpha
  modulesCount: 2
```

Module auto-update logic

> **i** ModuleRelease versions v1.0.0 and v1.1.1 are provided as examples.

1. **Module installation**. When a module is enabled (`enable module <module name>`), the latest available version from the selected stability channel is automatically downloaded and deployed to the cluster. For example, this could be ModuleRelease v1.0.0. The most recent version is used; older versions are not installed.
2. **Module disabling**. When a module is disabled (`disable module <module name>`):
   - The module stops receiving new releases.
   - The currently deployed version remains in the cluster with the `Deployed` status.
3. **Behavior on re-enabling**.

   If the module is re-enabled within 72 hours:

   - The previously deployed version (ModuleRelease v1.0.0) is used.
   - New releases are checked.
   - If available, they are downloaded (e.g., v1.1.0, v1.1.1).
   - The module is then updated according to the standard update rules (Update). More information

   If the module is re-enabled after 72 hours:

   - The old version is deleted (`delete ModuleRelease v1.0.0`).
   - Upon re-enabling, the latest available version is downloaded (e.g., v1.1.1).
   - The cycle starts again as if the module was enabled for the first time (see step 1).
4. **Behavior of a disabled module**. If a module is disabled, no new releases are downloaded. The previously deployed version (the last one that was enabled) is removed after 72 hours if the module is not re-enabled within that time.

Skipping intermediate releases (from-to)

The `from-to` mechanism allows you to skip step-by-step module updates. If the current installed module version (status `Deployed`) falls within the `from-to` range, DP skips intermediate releases and installs the latest available version within `to`.

To enable the mechanism, set transition rules in the module configuration (`module.yaml`). Example:

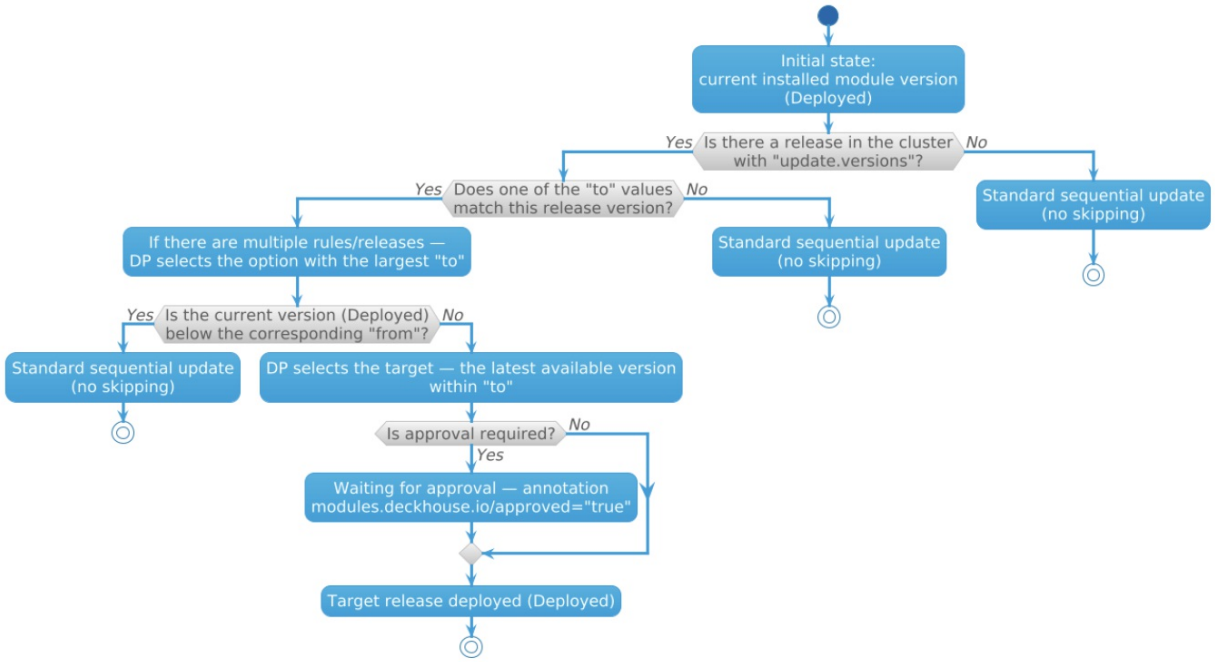
```
update:
  versions:
    - from: "1.67"
      to:   "1.75"
    - from: "1.99"
      to:   "2.0" # Transitions between major versions are specified in the X.Y format.
```

> **i** A constrained release is a module release whose `module.yaml` contains the `update.versions` section. The `from-to` mechanism works only with such releases.

Conditions for applying `from-to`:

- Target release version. The rule is read from the release you want to move to — the `to` value must match the version of that very release (constrained release).
- Current version threshold. The current installed module version (`Deployed`) is not lower than `from`. If the current version is lower than `from`, the rule does not apply — the update proceeds sequentially.
- If multiple releases match at the same time, the option with the largest `to` is chosen (the rules may reside in different ModuleRelease objects of the same module).
- If no release meets these conditions, the update proceeds as usual — without skipping intermediate versions.

If a release with update.versions appears in the cluster, DP does not require updating in order — such a release appears “as is” in the list, DP automatically selects a suitable option and, if necessary, waits for approval. You can immediately approve installation of the latest available version within `to`. After approval, intermediate releases between `from` and `to` get the `Skipped` status after reconciliation (not immediately); for some time between `Superseded` and `Deployed` there may be releases in the `Pending` status.



Check available releases (ModuleRelease) with:

```
d8 k get mr
```

Example output if the current module version is `0.3.33`, and `module.yaml` contains the rule `from: "0.3" → to: "0.7"`:

```
p-o-test-v0.1.0    Superseded
p-o-test-v0.2.22   Superseded
p-o-test-v0.3.33   Deployed
p-o-test-v0.4.1    Pending      Release is waiting for the
'modules.deckhouse.io/approved: "true"' annotation
p-o-test-v0.5.27   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.6.11   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.7.25   Pending      Release is waiting for the
'modules.deckhouse.io/approved: "true"' annotation
```

In this example output, the current installed version is `0.3.33` (status `Deployed`). According to the rule `from: 0.3 → to: 0.7`, the latest available version within `0.7` is selected for installation — `p-o-test-v0.7.25` (status `Pending`). After approval, versions `0.4.1`, `0.5.27`, and `0.6.11` change to `Skipped`, and `p-o-test-v0.7.25` becomes `Deployed`.

To approve the module release, apply the annotation:

```
d8 k annotate mr p-o-test-v0.7.25 modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve p-o-test v0.7.25
```

Example output after approval:

```
p-o-test-v0.4.1    Skipped
p-o-test-v0.5.27   Skipped
p-o-test-v0.6.11   Skipped
p-o-test-v0.7.25   Deployed
```

Whether an annotation is required depends on the module's update policy. More details — Module update policy.

Examples

Example 1. The target release contains the rule, and its version equals `to` — the transition is performed.

In this example, the `from-to` rule is defined in the **target** release, and the `to` value equals the version of that release. The current installed version (`Deployed`) is not lower than `from`, so the transition is possible. As soon as the release appears in the cluster, DP prepares the update. If the update policy requires approval, mark the release with the annotation. After approval, intermediate releases will be skipped and the target release becomes `Deployed`:

```
# Module version (Deployed): v1.67.23 (≥ 1.67) → the 'from' condition is satisfied.
metadata:
  name: module-test-v1.67.23
status:
  phase: Deployed

# Target release: the release version equals update.versions.to (1.75).
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"   # The update will be performed to this version after approval (if
required).
```

Likewise, if the current version is even higher than `from`, the transition is also performed:

```
# Module version (Deployed): v1.69.0  (≥ 1.67) → the 'from' condition is satisfied.
metadata:
  name: module-test-v1.69.0
status:
  phase: Deployed

# Target release: the release version equals update.versions.to (1.75).
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"  # The update will be performed to this version after approval (if
required).
```

Example 2. The current version is lower than `from` — the transition is not performed.

If the current installed version is lower than `from`, the rule does not apply. The update proceeds sequentially (without skipping):

```
# Module version (Deployed): v1.61.0  (< 1.67) → the 'from' condition is NOT satisfied.
metadata:
  name: module-test-v1.61.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"   # The from-to transition is not performed → sequential update.
```

Example 3. `to` does not match the version of the release that contains the rule — the transition is not performed.

A transition is possible only to a release where `to` equals the version of that release. If the rule is described in another release (for example, `to: "1.74"` is placed in release `v1.75.25`), it will not work — the update goes sequentially.

```
# Module version (Deployed): v1.67.0.
metadata:
  name: module-test-v1.67.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.74.0  # This release has no update.versions.

# The rule is written in another release (v1.75.25), and 'to' does not equal its
version.
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.74"   # 'to' does not equal this release version (1.75.25) → the rule is
ignored.
```

Module artifacts in the container registry

After a module has been built, its artifacts must be pushed to the container registry at a path that is the *source* path for pulling and running modules in DP. The path where module artifacts are pushed to the registry is specified in the ModuleSource resource.

Below is an example of the container image hierarchy after pushing the `module-1` and `modules-2` module artifacts into the registry:

```
registry.example.io
 modules-source
├─  module-1
│  ├─  v1.23.1
│  ├─  d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
│  ├─  e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
│  ├─  v1.23.2
│  └─  release
│     ├─  v1.23.1
│     ├─  v1.23.2
│     ├─  alpha
│     └─  beta
└─  module-2
   ├─  v0.30.147
   ├─  d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
   ├─  e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
   ├─  v0.31.1
   └─  release
      ├─  v0.30.147
      ├─  v0.31.1
      ├─  alpha
      └─  beta
```

> ⚠ The container registry must support a nested repository structure. See the requirements section for more details.

Below is a list of commands for working with the module source. The examples use the crane tool. Follow the instructions to install it. For macOS, use `brew`.

Print the list of modules in the module source

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>
```

An example:

```
$ crane ls registry.example.io/modules-source
module-1
module-2
```

Print the list of module images

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>
```

An example:

```
$ crane ls registry.example.io/modules-source/module-1
v1.23.1
d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
v1.23.2
```

In the example above, there are two module images and two application container images in `module-1`.

Print the list of files in the module image

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -tf -
```

An example:

```
crane export registry.example.io/modules-source/module-1:v1.23.1 - | tar -tf -
```

The output will be quite large.

Print the list of images of the module's application containers

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -Oxf - images_digests.json
```

An example:

```
$ crane export registry.example.io/modules-source/module-1:v1.23.1 -  | tar -Oxf - images_digests.json
{
  "backend": "sha256:fcb04a7fed2c2f8def941e34c0094f4f6973ea6012ccfe2deadb9a1032c1e4fb",
  "frontend": "sha256:f31f4b7da5faa5e320d3aad809563c6f5fcaa97b571fffa5c9cab103327cc0e8"
}
```

Configuring extra images

Modules can include additional images (such as vulnerability databases or other supplementary data) by adding an `extra_images.json` file. This file specifies extra images that need to be manually pushed to the registry and are separate from the main module images.

To view the extra images configuration:

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -Oxf - extra_images.json
```

An example of `extra_images.json` for neuvector vulnerability database:

```
{
  "scanner": 3
}
```

Important notes:

- Extra images must be manually pushed to the module registry under the `extra/` path.
- Use the `d8 mirror pull --only-extra-images` command to pull only extra images.
- Extra images are stored in the registry as `<module-name>/extra/<image-name>`.

Print the list of releases

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
```

An example:

```
$ crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
v1.23.1
v1.23.2
alpha
beta
```

In the example above, there are two releases in the container registry; two release channels, `alpha` and `beta`, are also used:

Print the version in use on the `alpha` release channel

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release:alpha - | tar -Oxf - version.json
```

An example:

```
$ crane export registry.example.io/modules-source/module-1/release:alpha - | tar -Oxf - version.json
{"version":"v1.23.2"}
```

Additional resources

- Global custom resources
- Module `deckhouse` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Marketplace

Marketplace is a package management system in Deckhouse Platform (DP). There are two types of packages:

- **Application**: A user workload deployed into a namespace.
- **Module**: A cluster capability extension.

Currently, only Application packages are supported. Module support is planned for a future DP version.

> Marketplace is available in DP starting from version 1.76.

To learn about connecting package repositories and managing repository scanning, refer to “Administration”.

To learn about installing and removing applications from Marketplace, as well as troubleshooting, refer to “Usage”.

Sections

The Marketplace documentation is divided into the following sections:

- The “Concepts” section describes package types, resource model, Application constraints, and application installation steps.
- The “Application development” section describes how to create an Application package from scratch: bootstrapping, project structure, `package.yaml`, local rendering and linting, build, CI/CD, and package artifacts in the container registry.
- The “Application settings” section describes the OpenAPI schemas of settings and values: validation, default values, and the extensions that control the settings form in the web interface.
- The “Templates” section describes the values available to Application templates, object naming, container images, Ingress, and application endpoints.
- The “Nelm annotations” section covers the full set of Nelm annotations used in Application templates to control deployment order, resource lifecycle, tracking, and logging.
- The “Hooks” section describes how to write Go hooks for Applications using the `ApplicationHookInput` type from module-sdk.
- The “Lifecycle and debugging” section describes how DP installs, updates, and deletes an Application, and which tools help to debug a package.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Concepts

This section describes package types, resource model, Application constraints, and application installation steps.

Package types

A package is an abstract unit that encompasses either an **Application** or a **Module**. The distinction is based on scope and purpose:

Characteristic	Module	Application
Purpose	Infrastructure extension for the cluster	User workload
Scope	Cluster-wide (one per cluster)	Namespaced (unlimited instances)
Multiple instances	No (1:1 with the cluster)	Yes (N instances in different namespaces)
Enabled by default	Can be enabled via bundle	Only by explicit user action
CRD creation	Allowed	Forbidden
Cluster-wide objects	Allowed	Forbidden

Resource model

Deckhouse Platform (DP) Marketplace uses five custom resources:

```
flowchart TD
    PR[PackageRepository] -->|Triggers| PRO[PackageRepositoryOperation]
    PR -->|Populates| APV[ApplicationPackageVersion]
    APV -->|Aggregated by| AP[ApplicationPackage]
    APV -->|Referenced by| APP[Application\nnamespace-scoped]
```

Resource	Short name	Scope	Role

Resource	Short name	Scope	Role
PackageRepository	—	Cluster	Connection to a container registry and scan schedule
PackageRepositoryOperation	`pro`	Cluster	Scan job that discovers versions
ApplicationPackageVersion	`apv`	Cluster	One per discovered package version. Carries metadata, OpenAPI schemas, and requirements
ApplicationPackage	`ap`	Cluster	Package summary: which repos have the package, which versions the release channels point to, how many instances use it
Application	—	Namespace	Installed instance. Drives Nelm deployment

ApplicationPackageVersion content

ApplicationPackageVersion contains package version metadata and information required to configure and deploy an application:

Content	Field
Localized package description (`en`/`ru`)	`status.packageMetadata.description`
Package maturity stage (`Preview`, `General Availability`, and others)	`status.packageMetadata.stage`
DP and Kubernetes version requirements and module dependencies (`mandatory`, `conditional`, `anyOf`, `noneOf`)	`status.packageMetadata.requirements`
Application removal confirmation settings. For details, refer to “Deletion confirmation”	`status.packageMetadata.disableOptions`
Package version changes from `changelog.yaml`	`status.packageMetadata.changelog`
OpenAPI v3 schema for validating `Application.spec.settings`	`status.packageSchemas.settingsSchema`
OpenAPI v3 schema for validating the final set of `values` passed to hooks and templates	`status.packageSchemas.valuesSchema`

Application installation steps

Installing an application from a package involves several stages, from connecting and scanning the package repository to creating an Application resource, deploying the application, and checking its status.

1. Administrator creates a PackageRepository.
2. DP creates a PackageRepositoryOperation automatically. The initial scanning is performed when the resource is created, and then repeated based on the `scanInterval` value.
3. The operation scans the repository and creates ApplicationPackageVersion objects for each discovered version.
4. User creates an Application in their namespace referencing package repository name (`packageRepositoryName`), application package name (`packageName`), and version (`packageVersion`).
5. DP validates `spec.settings` against the `settingsSchema` from the corresponding ApplicationPackageVersion.
6. Nelm deploys the Helm templates from the package bundle.
7. The `Installed` condition on the Application reflects deployment progress and is the only condition reported until the installation completes. Once it becomes `True`, the other conditions (`ConfigurationApplied`, `Scaled`, `Ready`, and so on) appear.

Application constraints

All constraints exist to enforce namespace isolation and prevent Applications from interfering with cluster-level resources.

Functional constraints

The following constraints apply:

1. **No CRD creation**: Application templates must not include CustomResourceDefinition objects.
2. **No cluster-wide objects**: All resources created by an Application must be namespaced.
3. **No cross-Application dependencies**: An Application can declare dependencies only on Modules (via `requirements.modules` in `package.yaml`), not on other Applications.
4. **Hooks are namespace-scoped**: Hooks must not read or write resources outside their own namespace.
5. **Manual install only**: Applications are never activated by default; installation requires explicit user action.

Naming constraints

The objects of an Application are named in the format `d8a-<INSTANCE_NAME>-<SUFFIX>` (for details, refer to “Templates”), so the instance name is a part of every object name. To keep the object names within the Kubernetes limits, the following naming constraints apply:

- **Application instance name** (`metadata.name`) can’t exceed **24 characters**. The validating webhook rejects an Application with a longer name.
- **Resource name suffix inside the Application** can’t exceed **23 characters** for StatefulSets, Jobs, and CronJobs, whose names must fit 52 characters (4 + 24 + 1 + 23 = 52), and at most **34 characters** for other objects, whose names must fit 63 characters.

For example, the `master` StatefulSet of the `redis-cache` instance (11 characters) is named `d8a-redis-cache-master` (22 characters).

Additional resources

- Global custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Application development

Prerequisites

Before you start, do the following:

1. Install the Deckhouse CLI (`d8`):

   ```
   sh -c "$(curl -fsSL https://raw.githubusercontent.com/deckhouse/deckhouse-cli/main/tools/install.sh)"
   ```

2. Log in to the container registry with your license key:

   ```
   d8 dk cr login -u license-token dev-registry.deckhouse.io --password <LICENSE_TOKEN>
   ```

Bootstrapping an Application package

To create a `<APPLICATION_NAME>/` directory in the current working directory with the package skeleton and initialize a Git repository with the first commit, run the command `d8 package bootstrap app <APPLICATION_NAME>` (you can use the short name `app` instead of `application`):

```
d8 package bootstrap application myapp --hooks
cd myapp
git remote add origin <GITLAB_REPO_URL>
git push --set-upstream origin main
```

Available options:

Option	Description

Option	Description
`--hooks`	Generate Go hooks: a hook example and a settings validation hook
`--werf`	Describe the image build in a `werf.inc.yaml` file instead of a `Dockerfile`
`--extended`	Add the `oss.yaml` file with the list of open source components used by the application
`-o, --output <OUTPUT_PATH>`	Path where the package will be created (default: `<CURRENT_WORKING_DIRECTORY>/<APPLICATION_NAME>`)

Project structure

In the generated project, the package manifest, schemas, templates, hooks, images, documentation, and continuous integration and continuous delivery (CI/CD) configuration are stored in separate directories and files.

```
myapp/
├── .gitignore
├── .gitlab-ci.yml             # CI/CD pipeline.
├── .pkglint.yaml              # Settings of the d8 package verify command.
├── changelog.yaml             # Changes in the package version.
├── docs/
│   ├── README.md              # Application documentation.
│   ├── README_RU.md
│   ├── CONFIGURATION.md
│   ├── CONFIGURATION_RU.md
│   └── icon.svg               # Application icon.
├── hooks/                     # Go hooks (--hooks).
│   ├── hooks.yaml             # Build instructions for the hooks binary.
│   └── batch/
│       ├── go.mod
│       ├── go.sum
│       ├── main.go
│       ├── settings/
│       │   └── check.go       # Settings validation hook.
│       └── triggers/
│           └── hook.go        # Hook example.
├── images/                    # Container images of the application.
│   └── echo/
│       └── Dockerfile         # werf.inc.yaml with --werf.
├── openapi/
│   ├── settings.yaml          # OpenAPI schema for Application.spec.settings.
│   ├── doc-ru-settings.yaml   # Russian descriptions of the settings.
│   └── values.yaml            # OpenAPI schema for Helm values.
├── oss.yaml                   # Open source components (--extended).
├── package.yaml               # Package manifest.
└── templates/                 # Helm templates.
    ├── _helpers/              # Template helpers.
    ├── deployment.yaml
    ├── pdb.yaml
    ├── registry-secret.yaml
    ├── service.yaml
    └── vpa.yaml
```

Don’t add `Chart.yaml` or `values.yaml` to the package root. The `d8 package build` command doesn’t include these files in the package bundle. Deckhouse Platform (DP) renders the templates as a chart without metadata, and default values are described in the OpenAPI schemas.

File package.yaml

The `package.yaml` file is the main manifest for an Application package and defines metadata, type, requirements, and compatibility.

Example `package.yaml` file:

```
apiVersion: v1
type: "Application"
name: redis
descriptions:
  ru: "<RU_DESCRIPTION>"
  en: "Redis — in-memory database"
# Injected automatically at build time.
version: "v1.0.1"
stage: "Preview"
category: "Databases"
# Environment requirements.
requirements:
  deckhouse:
    constraint: ">= 1.70"
  kubernetes:
    constraint: ">= 1.31"
  modules:
    mandatory:
      - name: cert-manager
        constraint: ">= 1.0.0"
```

Field reference:

Field	Required	Description
`name`	Yes	Unique package name
`descriptions`	Yes	Localized description for the catalog and user interface (UI) (`ru`, `en`)
`version`	Yes	Version in Semantic Versioning (SemVer) format; added automatically at build time
`type`	Yes	`Application` or `Module`
`stage`	Yes	Maturity stage (`Preview`, `General Availability`, etc.)
`category`	Yes	Category for catalog classification
`requirements.deckhouse`	No	Minimum DP version constraint
`requirements.kubernetes`	No	Minimum Kubernetes version constraint
`requirements.modules`	No	Module dependencies (SemVer constraints). For details, refer to “Requirements”
`disable`	No	Confirmation before the application is deleted. For details, refer to “Deletion confirmation”

When DP scans a repository, it publishes the description, stage, requirements, deletion confirmation, and the contents of `changelog.yaml` in the `status.packageMetadata` field of the ApplicationPackageVersion resource.

Requirements

The `requirements` section defines the conditions under which the application can be installed and run:

Requirement	Field
Supported DP version	`requirements.deckhouse.constraint`
Supported Kubernetes version	`requirements.kubernetes.constraint`
Modules that must be enabled. A version constraint (`constraint`) can be specified for each module	`requirements.modules.mandatory`

Requirement	Field
Optional modules. If such a module is enabled, its version must match the specified constraint (`constraint`)	`requirements.modules.conditional`
Groups of alternative modules. At least one module in each group must be enabled, and its version must match the corresponding `constraint`, if specified	`requirements.modules.anyOf`
Groups of incompatible modules. No module in such a group can be enabled. `constraint` specifies the incompatible versions; if no constraint is specified, all versions of the module are considered incompatible	`requirements.modules.noneOf`

Each `anyOf` and `noneOf` group has a unique `name`, which is used in error messages, an optional `description`, and a non-empty `modules` list. A module can be listed in only one of the `mandatory`, `conditional`, `anyOf`, and `noneOf` sections. Within `anyOf` or `noneOf`, the same module can be listed in several groups.

Example of a populated `requirements` section:

```
requirements:
  deckhouse:
    constraint: ">= 1.76"
  kubernetes:
    constraint: ">= 1.31"
  modules:
    mandatory:
      - name: cert-manager
    conditional:
      - name: prometheus
        constraint: ">= 1.60"
    anyOf:
      - name: storage
        description: "A storage for application data"
        modules:
          - name: sds-local-volume
          - name: csi-ceph
    noneOf:
      - name: local-path-storage
        description: "Local path volumes are not supported"
        modules:
          - name: local-path-provisioner
```

DP checks the requirements:

- **When an Application is created or changed**. If the requirements are not met, the request is rejected, and the message names the unmet requirement.
- **Before the installation**. DP installs the application only after the modules from `requirements.modules.mandatory` are enabled.
- **While the application is installed**. If the requirements stop being met, for example, a mandatory module is disabled, DP uninstalls the application (for details, refer to “Lifecycle and debugging”).

Module versions are compared with constraints without the pre-release and build metadata parts.

Deletion confirmation

The `disable` section defines a confirmation the user must give before the application is deleted in the web interface:

```
disable:
  confirmation: true
  messages:
    en: "Deleting the application deletes all the data stored in its volumes."
    ru: "<RU_MESSAGE>"
```

DP publishes the section in the `status.packageMetadata.disableOptions` field of the

ApplicationPackageVersion, and the web interface shows the message in the language of the user. DP doesn’t check the confirmation when an Application is deleted through the Kubernetes API.

File changelog.yaml

The `changelog.yaml` file describes the changes in the package version:

```
features:
  - "Added support for Redis 7.4."
fixes:
  - "Fixed the readiness probe of the replica."
```

DP publishes the contents of the file in the `status.packageMetadata.changelog` field of the ApplicationPackageVersion.

OpenAPI schemas

The `openapi/` directory defines two schemas:

- `settings.yaml`: Schema for `Application.spec.settings` (user-facing configuration). The legacy name `config-values.yaml` is also supported.
- `values.yaml`: Schema for the full set of Helm values.

The schemas, validation rules, and the extensions that control defaulting, immutability, and the settings form in the web interface are described in “Application settings”.

Templates and hooks

- The values available to templates and the rules for templates are described in “Templates”.
- Go hooks and the settings validation hook are described in “Hooks”.
- How DP installs, updates, and deletes an application and how to debug it is described in “Lifecycle and debugging”.

Local rendering

To render the templates of the package and print the resulting manifests, run the following command in the package directory:

```
d8 package render
```

Each object in the output is preceded by a comment with the name of its template file.

Available options:

Option	Description
`--file <FILE_NAME>`	Print only the objects rendered from the template with the specified file name
`--render-file <PATH>`	Write the manifests to a file, without the template file name comments
`-r, --remote <REPOSITORY>/<PACKAGE_NAME>:<PACKAGE_VERSION>`	Render a published package bundle instead of the local directory. A tag or a digest is required
`--remote-user`, `--remote-password`	Container registry credentials. They can also be set in the `PACKAGE_REMOTE_USER` and `PACKAGE_REMOTE_PASSWORD` environment variables

The command uses stub values instead of the values from a cluster: the `test` instance in the `default` namespace, the `dev` package version, and settings generated from `openapi/settings.yaml` (from `x-example`, `x-examples`, `enum`, or `default`). Image references are stubs keyed by the names of the image directories. To render the templates of an installed application with its actual values, use `deckhouse-controller packages render` (for details, refer to “Lifecycle and debugging”).

Linting

To check the structure, manifests, and schemas of the package, run the following command in the package directory:

```
d8 package verify
```

The command reports errors and warnings of the built-in rules and exits with an error if at least one error is

found. To get the full list of rules with descriptions, run `d8 package doc`.

The rules are grouped by linters:

Linter	What it checks
`package`	Required files (`changelog.yaml`, `docs/`), absence of build artifacts (`werf.yaml`, `.werf/`, `.helmignore`), and correctness of `requirements` in `package.yaml`
`openapi`	Types of `x-deckhouse-*` extension values, `x-deckhouse-ui-advanced` only on top-level settings, `enum` values in CamelCase, and a `doc-ru-*` file for each schema except `values.yaml`
`templates`	Object names (the `d8a-<INSTANCE_NAME>-` prefix, the suffix length of Job and CronJob names), absence of `metadata.namespace`, PodDisruptionBudget and VerticalPodAutoscaler objects for workloads, and named `targetPort` in Services. The templates are rendered with the `test` instance name in the `default` namespace
`docs`	Non-empty `docs/README.md`, a Russian version of every document, and no Cyrillic in English documents
`images`	Image directory names without `_` and the format of patch files
`icon`	Application icon `docs/icon.{png,webp,jpg,jpeg,svg}`: format, size up to 150 KB, and dimensions up to 300×300 pixels
`oss`	Format of `oss.yaml`, if the file exists

Available options:

`--hide-warnings`: Don’t show warnings. `--show-ignored`: Show the findings of ignored rules. `--lint-config <PATH>`: Path to the settings file.

To check a published package, run the following command:

```
d8 package verify remote <REPOSITORY> <PACKAGE_NAME>
```

By default, the command checks the bundle of the latest version. Use `--version <PACKAGE_VERSION>` to choose the version and `--release` to also check the version metadata image. The command uses the container registry credentials saved with `d8 dk cr login`.

File .pkglint.yaml

The `.pkglint.yaml` file lowers the severity of the rules. The command looks for it in the package directory and its parent directories (for `verify remote`, in the current directory).

Example:

```
version: "1"
static:          # d8 package verify.
  linters:
    templates:
      rules:
        vpa:
          impact: ignored
        service-port:
          impact: warn
remote:          # d8 package verify remote.
  bundle:
    linters:
      docs:
        impact: warn
```

The `impact` field takes the `error`, `warn`, and `ignored` values. The impact of a rule can’t be higher than the impact of its linter. The rules of the `package` and `openapi` linters can’t be configured, and the

`instance-prefix`, `instance-namespace`, and `job-name` rules follow the impact of the `templates` linter.

Local build

To build the package and publish it to a container registry, run:

```
d8 package build -v v0.0.1 -r dev-registry.deckhouse.io/deckhouse/packages
```

For local development, use the `payload-registry` module as your own container registry.

When using the command, consider the following:

- Specify the root path of the packages in `-r`, the same as the `spec.registry.repo` field of the PackageRepository resource. The command appends the package name to the path itself.
- Specify the version in the `vMAJOR.MINOR.PATCH` format. DP finds only the versions in this format when it scans the repository.
- The images are built with werf (`d8 delivery-kit`) for the `linux/amd64` platform, so the package directory must be a Git repository. Uncommitted changes are also included in the build.
- The `images/` directory must contain at least one image.
- If the version already exists in the registry, the command exits without building. To rebuild the version, use `-f`.

Available options:

Option	Environment variable	Description
`-v, --version`	—	Package version (required)
`-r, --repo`	`PACKAGE_BUILD_REPOSITORY`	Root path of the packages in the container registry. Without it, the package is only built locally
`-u, --user`, `-t, --token`	`PACKAGE_BUILD_REPOSITORY_USER`, `PACKAGE_BUILD_REPOSITORY_TOKEN`	Container registry credentials
`--final-repo`, `--final-user`, `--final-token`	`PACKAGE_BUILD_FINAL_REPOSITORY`, `PACKAGE_BUILD_FINAL_REPOSITORY_USER`, `PACKAGE BUILD_FINAL_REPOSITORY_TOKEN`	Container registry to publish the package to (path and credentials), if it differs from the build registry
`-f, --force`	—	Rebuild and publish a version that already exists in the registry
`--insecure`	`PACKAGE_BUILD_INSECURE`	Allow HTTP and skip the verification of the TLS certificates of the container registries
`--sign`, `--sign-cert`, `--sign-key`	`PACKAGE BUILD SIGN CERT`, `PACKAGE_BUILD_SIGN_KEY`	Sign the images with the specified certificate and key

CI/CD setup

The CI/CD pipeline publishes package releases to the container registry. To publish a release, configure the container registry credentials, then create a Git tag in SemVer format and push it to the repository.

Environment variables

The pipeline uses the following variables to authenticate to the container registry:

Variable	Description
`PACKAGES_REGISTRY_LOGIN`	Container registry username for publishing
`PACKAGES_REGISTRY_PASSWORD`	Container registry password or token

Triggering a release

The pipeline is triggered by a Git tag in SemVer format:

```
git tag v0.1.0
git push origin v0.1.0
```

The pipeline builds the package and pushes it to the container registry. Once the pipeline completes, the package version is available for scanning via PackageRepository.

Package artifacts in the container registry

The package and related data are published to a container registry. The package bundle, container images, and version metadata are stored at separate paths.

Path	Description
`<REPOSITORY>:<PACKAGE_NAME>`	Package name tag — used to list packages
`<REPOSITORY>/<PACKAGE_NAME>:<PACKAGE_VERSION>`	Bundle — contains templates, `openapi/`, `hooks/`
`<REPOSITORY>/<PACKAGE_NAME>@<DIGEST>`	Container images of the application. Templates reference them by digest (for details, refer to “Templates”)
`<REPOSITORY>/<PACKAGE_NAME>/version:<PACKAGE_VERSION>`	Version metadata
`<REPOSITORY>/<PACKAGE NAME>/release-channel:<RELEASE_CHANNEL>`	Version recommended for a release channel

Here `<REPOSITORY>` is the root path of the packages, for example, `registry.deckhouse.io/deckhouse/<EDITION>/packages`.

Bundle contents

The main bundle image (`<PACKAGE_NAME>:<PACKAGE_VERSION>`) contains:

```
├── package.yaml         # Package manifest with the version.
├── images_digests.json  # Digests of the container images.
├── openapi/             # Settings and values schemas.
├── templates/           # Helm templates.
├── charts/              # Helm subcharts, if any.
├── hooks/               # Hooks binary.
├── docs/                # Documentation and icon.
├── changelog.yaml       # Release notes.
└── oss.yaml             # Open source components, if any.
```

Version metadata image contents

The metadata image (`<PACKAGE_NAME>/version:<PACKAGE_VERSION>`) contains:

```
├── package.yaml       # Package manifest.
├── version.json       # SemVer version.
├── changelog.yaml     # Release notes.
├── openapi/           # Settings and values schemas.
└── docs/              # Documentation and icon.
```

DP reads the version metadata when it scans the repository and creates ApplicationPackageVersion objects from it.

Release channels

A release channel is a tag of the `<PACKAGE_NAME>/release-channel` path that points to a copy of the version metadata image of the recommended version. DP recognizes the `alpha`, `beta`, `early-access`, `stable`, `rock-solid`, and `lts` channels. The `d8 package build` command doesn’t create release channel tags; they are published by the CI/CD pipeline.

When DP scans a repository, it records the versions the channels point to in the `status.releaseChannels` field of the ApplicationPackage resource:

```
status:
  releaseChannels:
    my-registry:      # PackageRepository name.
      alpha: v0.2.0
      stable: v0.1.21
```

The channels are informational: the version of an application is set in `spec.packageVersion` of the Application resource and changes only when the user changes it.

Additional resources

- Global custom resources
- Module `payload-registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Application settings

The user configures an application instance in the `spec.settings` field of the Application resource. The package describes the allowed settings with an OpenAPI schema. Deckhouse Platform (DP) validates the settings against this schema, applies default values, and passes the result to templates and hooks. The DP web interface builds the application settings form from the same schema.

Schema files

The `openapi/` directory of the package contains two schemas:

- `settings.yaml`: Schema of `Application.spec.settings`, that is, of the settings the user can change. The legacy file name `config-values.yaml` is supported for backward compatibility: DP reads it only if `settings.yaml` is absent.
- `values.yaml`: Schema of the full set of values available to templates as `.Values`, including internal values set by hooks. In this schema, the package also lists the kinds of orphan resources that DP deletes together with the application.
- `doc-ru-settings.yaml`: Russian descriptions of the settings. The `d8 package verify` command requires a `doc-ru-<NAME>.yaml` file for every schema file except `values.yaml`.

The settings are located at the root of the values, so the values schema must declare them too. To include all settings in the values schema without duplicating them, add the `x-extend` extension to `openapi/values.yaml`. Without it, the values fail the validation when a hook changes them.

Example `openapi/values.yaml`:

```
x-extend:
  schema: settings.yaml
type: object
properties:
  internal:
    type: object
    default: {}
    properties:
      adminPassword:
        type: string
```

When DP scans a repository, it publishes both schemas in the `status.packageSchemas` field of the ApplicationPackageVersion resource. The web interface and the Application validating webhook use the published schemas. Only standard OpenAPI keywords and the `x-deckhouse-*` extensions described on this page are published. Other `x-*` extensions are removed from the published schema.

How DP builds values

DP builds values from the following sources, in the order listed:

1. Default values from `openapi/settings.yaml`.
2. Project defaults for fields with the `x-deckhouse-grantable-resource` extension.
3. User settings from `Application.spec.settings`. They override the values from the previous sources.
4. Default values from `openapi/values.yaml` for the fields that are still not set.
5. Values set by hooks.

Templates get the result as `.Values`. The effective settings (sources 1–3) are also available to templates as `.Application.Settings`. After the settings are applied successfully, DP saves the effective settings in the `status.lastAppliedConfiguration` field of the Application.

Validation

Undeclared fields

DP adds `additionalProperties: false` to every object of both schemas that doesn’t define `additionalProperties` explicitly. As a result, a setting or a value that is not declared in the schema is rejected.

To allow arbitrary keys in an object, for example, in a map of labels, define `additionalProperties` explicitly:

```
type: object
properties:
  podLabels:
    type: object
    additionalProperties:
      type: string
```

⚠ Fields declared only inside `oneOf`, `anyOf`, or `allOf` branches are also rejected. Declare every field of an object in its `properties`, and use the branches only to restrict combinations of values.

When settings are validated

When	What DP checks	Result of a failed check
An Application is created or changed	The settings schema published in ApplicationPackageVersion, including CEL rules that don’t reference `oldSelf`, and fields with `x-deckhouse-immutable`	The request is rejected
The settings of an installed application are changed (`spec.packageVersion` matches the installed version)	Additionally: CEL rules that reference `oldSelf`, availability of grantable resources, and the settings validation hook	The request is rejected
DP applies the settings	The same as in the previous row	The settings are not applied, the error is reported in the Application conditions
A hook changes values	The `openapi/values.yaml` schema, including CEL rules	The hook fails

CEL rules

Both schemas support the `x-deckhouse-validations` extension with rules in the Common Expression Language (CEL). The syntax and the semantics are the same as in module schemas: a rule consists of `expression` and `message`, the value at the current level is available as `self`, and the previous value is available as `oldSelf`.

Example `openapi/settings.yaml` with a rule that checks two fields together:

```
type: object
properties:
  replicas:
    type: integer
    default: 1
  maxReplicas:
    type: integer
    default: 3
x-deckhouse-validations:
  - expression: "self.replicas <= self.maxReplicas"
    message: "replicas must not exceed maxReplicas"
```

Rules that reference `oldSelf` compare the new settings with the previously applied ones. Such rules are checked only when the settings of an installed application change: they are skipped on installation and when `spec.packageVersion` changes. To forbid changing a field after installation, use `x-deckhouse-immutable`, which is checked on every update.

Defaulting a grantable cluster-wide resource value

A `settings` field of type `string` can be bound to a grantable cluster-wide resource managed by the `multitenancy-manager` (for example, a StorageClass).

When the field is bound and the user leaves it empty, the resource name configured as the project default is injected into `values`. When the user provides a value, it is checked against the resources available to the project. A value that is not in this list is rejected.

To bind the field to a cluster-wide resource, add the `x-deckhouse-grantable-resource` extension and specify the name of the resource available to the project through a grant (`AvailableClusterResource` or `GrantableClusterResourceDefinition`), for example, `storageclasses`.

> **i** The resource group, version, and kind (GVK) are defined by the grant. You do not need to specify them in `openapi/settings.yaml`.

Example `openapi/settings.yaml` with `x-deckhouse-grantable-resource`:

```
type: object
properties:
  storageClass:
    type: string
    x-deckhouse-grantable-resource: storageclasses
  postgres:
    type: object
    properties:
      storageClass:
        type: string
        x-deckhouse-grantable-resource: postgresclasses
```

Behavior:

- The default is resolved per project from the AvailableClusterResource in the Application’s namespace, so different projects can receive different defaults.
- An explicitly specified user value has higher priority than the project default.
- If the custom resource definition (CRD) is absent, no catalog exists for the project, or the catalog has no default value, the field remains unchanged. No value is injected or validated.

Immutable fields

Some settings should not be changed after the application configuration is applied. For example, changing a `storageClass` after volumes have been created either has no effect or can cause application failures. To prevent changes to the value of such a field after the application configuration has been successfully applied, add the `x-deckhouse-immutable: true` extension to it.

Example `openapi/settings.yaml` with `x-deckhouse-immutable`:

```
type: object
properties:
  storageClass:
    type: string
    default: default
    x-deckhouse-immutable: true
  postgres:
    type: object
    properties:
      storageClass:
        type: string
        x-deckhouse-immutable: true
      volumeSize:
        type: string
```

Behavior:

- The extension is effective only when set to `true`. Any other value is ignored.
- When `x-deckhouse-immutable` is added to an object, the entire object becomes immutable. After the application configuration has been successfully applied for the first time, changing any nested field is rejected, even if `x-deckhouse-immutable` is not set on that field. Set the extension on an object only

when the entire object must be immutable. To make only one field immutable, add `x-deckhouse-immutable` directly to it, as with `postgres.storageClass` in the example above. In this case, the restriction does not apply to `postgres.volumeSize`, and its value can be changed.
- If an update changes an immutable value, the validating webhook rejects it and reports the field name.
- In the web interface, the value of a field with the extension can be set when installing the application. In the edit form of an installed application, the field is read-only.
- Comparison uses the configuration that was actually applied, after schema defaults are applied. Therefore, a field with the extension can be omitted from the manifest only if its `default` matches the value that has already been applied. If there is no default value or it differs from the applied value, the change is rejected.
- The extension doesn’t apply to array elements and map entries that an update adds: a new element has no previous value to compare with.

i If an entire object is removed from the manifest, default values are not applied to its nested fields. Therefore, removing an object that contains fields using `x-deckhouse-immutable` can cause frozen values to be lost and is rejected.

Web interface form

The web interface builds the application settings form from `openapi/settings.yaml`. The following extensions control how fields are displayed:

Extension	Value	Effect
`x-deckhouse-ui-order`	Integer	Display order of the field. Fields with lower values are shown first
`x-deckhouse-ui-group`	String	The field is shown in the root-level group with this name. The path of the field in the settings doesn’t change
`x-deckhouse-ui-advanced`	`true`	The field is hidden behind the advanced settings toggle. The extension is taken into account only for top-level fields
`x-deckhouse-ui-validation-message`	String	The message shown under the field instead of the schema validation error, for example, instead of a regular expression that the value doesn’t match
`x-deckhouse-ui-resource-name`	Object with the `apiVersion` and `kind` fields and an optional `labelSelector` field	For a field of type `string`: a drop-down list with the names of resources of the specified kind from the application namespace. `labelSelector` is a standard Kubernetes label selector that narrows the list
`x-deckhouse-ui-text-area`	`true`	For a field of type `string`: a multi-line text field instead of a single-line one. See “Multi-line and masked fields”
`x-deckhouse-ui-secure-field`	`true`	For a field of type `string`, `integer`, or `number`: the input is masked like a password. The value is stored in the settings as is. See “Multi-line and masked fields”
`x-deckhouse-ui-sizing-memory`	Object with the `defaultUnit` field	For a field of type `integer` that stores a memory size in bytes: the size is entered with a unit, for example, `512Mi`. See “Memory size fields”
`x-deckhouse-enum-switch-settings`	List, declared on a `oneOf` branch	Confirmation dialogs shown before switching to another branch. See the next section

Example `openapi/settings.yaml` with form extensions:

```
type: object
properties:
  replicas:
    type: integer
    default: 1
    x-deckhouse-ui-order: 10
  domain:
    type: string
    pattern: '^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'
    x-deckhouse-ui-order: 20
    x-deckhouse-ui-validation-message: "Enter a domain name, for example,
app.example.com."
  tlsSecretName:
    type: string
    x-deckhouse-ui-group: TLS
    x-deckhouse-ui-resource-name:
      apiVersion: v1
      kind: Secret
      labelSelector:
        matchLabels:
          example.com/tls: "true"
  logLevel:
    type: string
    enum: [Info, Debug]
    default: Info
    x-deckhouse-ui-advanced: true
```

Confirmation before switching a mode

Some settings switches can’t be reverted without data loss, for example, switching the storage of an application from an in-cluster one to an external one. To make the web interface ask for a confirmation before such a switch, add the `x-deckhouse-enum-switch-settings` extension to a `oneOf` branch. The confirmation is shown when the branch discriminator (the field whose `enum` value selects the branch) changes from a value of this branch to one of the target values.

Each list item contains the following fields:

- `to`: Target discriminator values the confirmation applies to. The values are compared as strings, so boolean discriminators are specified as `"true"` and `"false"`.
- `impact`: Severity of the switch: `info` (default), `warning`, or `destructive`. The web interface applies a `destructive` switch only after an explicit acknowledgement and records it in the `applications.deckhouse.io/allow-destructive-settings-switch` annotation of the Application.
- `messages`: Dialog text in Markdown by language code (`en`, `ru`).

Example `openapi/settings.yaml` with `x-deckhouse-enum-switch-settings`:

```
type: object
properties:
  storage:
    type: object
    default: {}
    properties:
      mode:
        type: string
        enum: [Internal, External]
        default: Internal
      size:
        type: string
      url:
        type: string
    oneOf:
      - properties:
          mode:
            enum: [Internal]
        x-deckhouse-enum-switch-settings:
          - to: [External]
            impact: destructive
            messages:
              en: "Switching to external storage **deletes the data** stored in the
cluster."
              ru: "<RU_MESSAGE>"
      - properties:
          mode:
            enum: [External]
        required: [url]
```

Multi-line and masked fields

The `x-deckhouse-ui-text-area` and `x-deckhouse-ui-secure-field` extensions change only how the web interface shows the input of a field:

- `x-deckhouse-ui-text-area: true`: A field of type `string` is shown as a multi-line text field, for example, for a configuration fragment or a certificate.
- `x-deckhouse-ui-secure-field: true`: The input of a field of type `string`, `integer`, or `number` is masked like a password.

A field of type `string` with both extensions is a multi-line text field with masked input. Fields with `enum` ignore both extensions. The extensions take effect only with the boolean `true` value. If the value is a string, for example, `"true"`, Deckhouse Platform can’t load the package version.

Example `openapi/settings.yaml` with `x-deckhouse-ui-text-area` and `x-deckhouse-ui-secure-field`:

```
type: object
properties:
  adminPassword:
    type: string
    minLength: 12
    x-deckhouse-ui-secure-field: true
  guestPin:
    type: integer
    minimum: 100000
    maximum: 999999
    x-deckhouse-ui-secure-field: true
  welcomeMessage:
    type: string
    x-deckhouse-ui-text-area: true
  sshPrivateKey:
    type: string
    x-deckhouse-ui-text-area: true
    x-deckhouse-ui-secure-field: true
```

⚠ The `x-deckhouse-ui-secure-field` extension doesn't protect the value. The value is stored in the `spec.settings` field of the Application as is and is available to everyone who can read the Application. To pass a secret, let the user create a Secret and select it in the form with the `x-deckhouse-ui-resource-name` extension.

Memory size fields

To let the user enter a memory size with a unit, for example, a cache limit, add the `x-deckhouse-ui-sizing-memory` extension to the field. The field must be of type `integer` and store the size in bytes. The `default`, `minimum`, and `maximum` values of the field are specified in bytes too.

The extension value is an object with the `defaultUnit` field. This field sets the unit in which the web interface shows the value when the form is opened: `Ki`, `Mi`, `Gi`, or `Ti`. These are the binary suffixes that Kubernetes uses for resource quantities: 1 Ki is 1024 bytes. If the extension value is not an object, for example, `x-deckhouse-ui-sizing-memory: Mi`, Deckhouse Platform can't load the package version.

The field works as follows:

- The user enters a size with a suffix, for example, `512Mi` or `2Gi`. A number without a suffix is a number of bytes. The web interface saves the size to the settings in bytes: `512Mi` becomes `536870912`. Fractional bytes are rounded.
- When the form is opened, the field shows the value in `defaultUnit` if the value is a whole number in this unit. Otherwise, the field shows the value in the largest unit in which it is a whole number: for example, with `defaultUnit: Gi`, `1610612736` is shown as `1536Mi`. A value that is not a multiple of 1024 is shown in bytes.
- For fields of other types, fields with `enum`, and `defaultUnit` values not listed above, the extension is ignored, and the field is shown as a regular number field.

Example `openapi/settings.yaml` with `x-deckhouse-ui-sizing-memory`:

```
type: object
properties:
  cache:
    type: object
    default: {}
    properties:
      maxMemory:
        type: integer
        default: 536870912
        minimum: 67108864
        x-deckhouse-ui-sizing-memory:
          defaultUnit: Mi
```

In the templates, the size is a number. Print it with the `int64` function: otherwise, a large number is printed in exponential notation, for example, `1.073741824e+09`:

```
env:
  - name: CACHE_MAX_MEMORY
    value: "{{ .Values.cache.maxMemory | int64 }}"
```

Additional resources

- Global custom resources
- Module `multitenancy-manager` documentation

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Templates

The `templates/` directory of the package contains Helm templates. Deckhouse Platform (DP) renders them and applies the result with Nelm, so the templates can use Nelm annotations to control the deployment order, resource lifecycle, and readiness tracking. This page describes the data DP passes to the templates

and the rules the templates must follow.

Template values

DP renders the templates with three value roots:

Root	Contents
`.Values`	Application values: settings with default values, internal values, and values set by hooks (for details, refer to “How DP builds values”). The values are not nested under the package name: a setting is available as `.Values.<SETTING_NAME>`
`.Application`	Instance and package parameters
`.Platform`	DP global parameters (read-only)

Root .Application

The `.Application` root contains instance and package parameters.

Path	Description
`.Application.Instance.Name`	Instance name, that is, the name of the Application resource
`.Application.Instance.Namespace`	Namespace the application is installed into
`.Application.Package.Name`	Package name (the `name` field of `package.yaml`)
`.Application.Package.Version`	Package version
`.Application.Package.Images`	Container images of the package: a map of image names to image references with a digest (for details, refer to “Container images”)
`.Application.Package.Registry`	Access parameters of the repository the package is installed from (for details, refer to “Container images”)
`.Application.Settings`	Effective settings: `Application.spec.settings` with default values applied

Root .Platform

`.Platform` contains the DP global parameters. The following parameters are useful in application templates:

Path	Description
`.Platform.applications.publicDomainTemplate`	DNS name template for application Ingresses (for details, refer to “Ingress and HTTPS”)
`.Platform.applications.ingressClass`	IngressClass for application Ingresses. Default: `nginx`
`.Platform.applications.https.mode`	HTTPS mode for applications: `CertManager` (default), `CustomCertificate`, `Disabled`, or `OnlyInURI`
`.Platform.applications.https.certManager.clusterIssuerName`	The ClusterIssuer of the `cert-manager` module for application certificates. Default: `letsencrypt`
`.Platform.discovery.clusterDomain`	Cluster domain, for example, `cluster.local`
`.Platform.discovery.kubernetesVersion`	Kubernetes version of the cluster
`.Platform.deckhouseVersion`	DP version
`.Platform.deckhouseEdition`	DP edition

The administrator configures the `.Platform.applications` parameters for all applications of the cluster in the `applications` section of the global settings.

Object names

Name every object in the templates as `d8a-<INSTANCE_NAME>-<SUFFIX>`:

```
metadata:
  name: d8a-{{ .Application.Instance.Name }}-server
```

The `d8 package verify` command reports an error (the `instance-prefix` rule) for every rendered object whose name doesn't start with `d8a-{{ .Application.Instance.Name }}-`. For Job and CronJob objects, the suffix after this prefix must be at most 23 characters long (the `job-name` rule).

The prefix serves the following purposes:

- Several instances of the package can be installed into the same namespace without name conflicts.
- The `d8a-` prefix is reserved for applications: the `d8a-prefix.deckhouse.io` admission policy forbids users to create, change, and delete objects whose names start with `d8a-`. Deleting Pods and changing Secrets with the `packages.deckhouse.io/initial-secret` label are allowed (for details, refer to "Credentials in the web interface").
- DP adds the same prefix to the names of objects that hooks create, patch, or delete. Therefore, a hook can address an object rendered by the templates by its suffix only (for details, refer to "Hooks").

Don't use `.Release.Name` in object names: the Helm release of an application is named `<NAMESPACE>.<INSTANCE_NAME>`, and the dot is not allowed in the names of many resource kinds.

Keep suffixes short: the instance name can be up to 24 characters long, and the full object name must fit the Kubernetes limits (for details, refer to "Naming constraints").

Labels and object protection

DP adds the following labels to the metadata of every rendered object:

Label	Value
`heritage`	`deckhouse`
`packages.deckhouse.io/package`	Package name
`packages.deckhouse.io/instance`	Instance name
`health.deckhouse.io/package`	`<NAMESPACE>.<INSTANCE_NAME>`

The labels are added only to the metadata of the rendered objects, not to Pod templates. Define the labels used in Pod selectors in the templates yourself.

When the Application is deleted, DP finds the objects that the instance has left in the namespace by the `packages.deckhouse.io/instance` label (for details, refer to "Orphan resources").

Objects with the `heritage: deckhouse` label are protected by the `label-objects.deckhouse.io` admission policy, and objects with the `d8a-` name prefix are protected by the `d8a-prefix.deckhouse.io` policy. As a result:

- Users can't change or delete objects rendered by the templates. The exception is Secrets with the `packages.deckhouse.io/initial-secret` label: users can change them, but not create or delete them (for details, refer to "Credentials in the web interface"). To change them manually, for example, when debugging, switch the application to maintenance mode.
- Workloads of the application can't change objects rendered by the templates either.
- The `d8a-prefix.deckhouse.io` policy doesn't apply to service accounts whose names start with `d8a-`. Workloads that run under such service accounts can manage objects with the `d8a-` prefix they create themselves, while these objects stay protected from users.

Container images

Reference the container images of the package through `.Application.Package.Images`. The key is the name of the image directory in `images/` converted to camelCase (for example, `images/my-server` becomes `myServer`). The value is a full image reference with a digest, for example, `registry.example.com/packages/myapp@sha256:...`:

```
containers:
  - name: server
    image: {{ index .Application.Package.Images "server" }}
```

The `d8 package render` command uses the directory names as is, so use single-word names for image directories: then the same key works both in local rendering and in the cluster.

To pull images from the repository, create an image pull Secret from `.Application.Package.Registry.dockercfg` and reference it in `imagePullSecrets`:

```
apiVersion: v1
kind: Secret
metadata:
  name: d8a-{{ .Application.Instance.Name }}-registrysecret
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: {{ .Application.Package.Registry.dockercfg }}
```

The section `.Application.Package.Registry` contains the following fields:

Field	Description
`name`	Name of the PackageRepository resource
`repository`	Repository address from the PackageRepository resource
`dockercfg`	Base64-encoded Docker configuration for accessing the repository. If the PackageRepository uses `login` and `password`, DP builds the configuration from them
`scheme`	Protocol for accessing the repository as specified in the PackageRepository resource, for example, `HTTPS`. DP passes the value as is: it can be in any case or empty, which means HTTPS
`ca`	CA certificate of the repository, if specified

Ingress and HTTPS

Build host names and TLS settings of application Ingresses from the `.Platform.applications` parameters:

- `publicDomainTemplate`: DNS name template with two `%s` placeholders. The first one is replaced with the instance name, the second one with the namespace. For example, with the `%s.%s.apps.example.com` template, the `grafana` instance in the `monitoring` namespace gets the `grafana.monitoring.apps.example.com` host name. If the parameter is not set, don’t create Ingresses.
- `ingressClass`: IngressClass of the Ingresses.
- `https.mode`: HTTPS mode. In the `CertManager` mode, request a certificate from the ClusterIssuer specified in `https.certManager.clusterIssuerName`. In the `Disabled` and `OnlyInURI` modes, don’t configure TLS on the Ingress.

Example Ingress:

```
{{- $applications := .Platform.applications }}
{{- if $applications.publicDomainTemplate }}
{{- $host := printf $applications.publicDomainTemplate .Application.Instance.Name
.Application.Instance.Namespace }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: d8a-{{ .Application.Instance.Name }}-web
  annotations:
    packages.deckhouse.io/application-endpoint-description: "Web UI"
    {{- if eq $applications.https.mode "CertManager" }}
    cert-manager.io/cluster-issuer: {{ $applications.https.certManager.clusterIssuerName
}}
    {{- end }}
spec:
  ingressClassName: {{ $applications.ingressClass }}
  {{- if eq $applications.https.mode "CertManager" }}
  tls:
    - hosts:
       - {{ $host }}
      secretName: d8a-{{ .Application.Instance.Name }}-web-tls
  {{- end }}
  rules:
    - host: {{ $host }}
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: d8a-{{ .Application.Instance.Name }}-server
                port:
                  name: http
{{- end }}
```

The package skeleton created by `d8 package bootstrap app` contains helpers for these parameters in `templates/_helpers/`:

- `public domain`
- `ingress_class`
- `https mode`
- `https_ingress_tls_enabled`
- `https_cert_manager_cluster_issuer_name`

Application endpoints

To show the addresses of an application in the `status.urls` field of the Application (for example, the web interface shows them as links to the application), add the `packages.deckhouse.io/application-endpoint-description` annotation to an Ingress, to an HTTPRoute if the application is published through the `alb` module, or to a ServiceAccount:

- The annotation value is the endpoint description. The `"true"` value adds the endpoint without a description, and the `"false"` value excludes the object.
- Only objects rendered by the templates are taken into account: Ingress objects of the `networking.k8s.io` API group, HTTPRoute objects of the `gateway.networking.k8s.io` API group, and ServiceAccount objects (`apiVersion: v1`).
- Deckhouse Platform updates `status.urls` after every successful apply of the templates.

For an Ingress, Deckhouse Platform builds a URL for every `host` and `path` pair of `spec.rules`. The scheme is `https` if the host is listed in `spec.tls`, and `http` otherwise. A rule without paths gives the `<SCHEME>://<HOST>/` URL. Rules without `host` are skipped.

Example status of an Application with the Ingress from the previous section:

```
status:
  urls:
    - url: https://myapp.my-namespace.apps.example.com/
      description: Web UI
```

HTTPRoute endpoints

For an HTTPRoute, Deckhouse Platform builds a URL for every host name of `spec.hostnames` and every path of `spec.rules`:

- The path is the value of a `PathPrefix` or `Exact` match. A rule without matches and a match without a path give the `/` path. Matches of the `RegularExpression` type are skipped.
- If `spec.hostnames` is empty, Deckhouse Platform takes the host names of the listeners the HTTPRoute is attached to.
- The scheme is `https` if the HTTPRoute is attached to an `HTTPS` listener that serves the host name, and `http` otherwise. Deckhouse Platform takes into account only the listeners of ListenerSet objects rendered by the templates. For an HTTPRoute attached directly to a Gateway or to a ListenerSet created outside the templates, the scheme is `http`.

Example ListenerSet and HTTPRoute:

```
{{- $applications := .Platform.applications }}
{{- if $applications.publicDomainTemplate }}
{{- $host := printf $applications.publicDomainTemplate .Application.Instance.Name
.Application.Instance.Namespace }}
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: d8a-{{ .Application.Instance.Name }}-web
spec:
  parentRef:
    name: <GATEWAY_NAME>
    namespace: <GATEWAY_NAMESPACE>
  listeners:
    - name: web
      port: 443
      protocol: HTTPS
      hostname: {{ $host }}
      tls:
        mode: Terminate
        certificateRefs:
          - name: d8a-{{ .Application.Instance.Name }}-web-tls
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: d8a-{{ .Application.Instance.Name }}-web
  annotations:
    packages.deckhouse.io/application-endpoint-description: "Web UI"
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: ListenerSet
      name: d8a-{{ .Application.Instance.Name }}-web
      sectionName: web
  hostnames:
    - {{ $host }}
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /
      backendRefs:
        - name: d8a-{{ .Application.Instance.Name }}-server
          port: 80
{{- end }}
```

where:

- `<GATEWAY_NAME>` and `<GATEWAY_NAMESPACE>` are the name and namespace of the Gateway object that the cluster administrator provides for publishing applications;
- `d8a-{{ .Application.Instance.Name }}-web-tls` is a Secret with the TLS certificate for the host

name, for example, issued by cert-manager. For details, refer to the `alb` module documentation.

For this HTTPRoute, Deckhouse Platform adds the same endpoint to `status.urls` as for the example Ingress: `https://myapp.my-namespace.apps.example.com/` with the `Web UI` description.

ServiceAccount endpoints

If the application gives users a ServiceAccount token for accessing the Kubernetes API, add the annotation to this ServiceAccount. For a ServiceAccount, Deckhouse Platform adds the in-cluster address of the Kubernetes API, `https://kubernetes.default.svc/`, to `status.urls`. This address is the same for all ServiceAccounts and is reachable only from inside the cluster.

Example ServiceAccount:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: d8a-{{ .Application.Instance.Name }}-api
  annotations:
    packages.deckhouse.io/application-endpoint-description: "Kubernetes API"
```

For this ServiceAccount, Deckhouse Platform adds the `https://kubernetes.default.svc/` endpoint with the `Kubernetes API` description to `status.urls`. To show the ServiceAccount token next to this endpoint in the web interface, bind a Secret with the token to the ServiceAccount (for details, refer to “ServiceAccount token”).

Credentials in the web interface

The web interface can show the credentials for signing in next to an application endpoint, for example, the login and password of the administrator account that the application creates during installation. To do this, add a Secret with the login and password to the templates and bind this Secret to the endpoint:

```
apiVersion: v1
kind: Secret
metadata:
  name: d8a-{{ .Application.Instance.Name }}-web-credentials
  labels:
    packages.deckhouse.io/application-instance: {{ .Application.Instance.Name }}
    packages.deckhouse.io/application-ingress: d8a-{{ .Application.Instance.Name }}-web
    packages.deckhouse.io/initial-secret: "true"
type: Opaque
data:
  login: {{ "admin" | b64enc }}
  password: {{ .Values.internal.adminPassword | b64enc }}
```

The web interface finds the credentials as follows:

- The `packages.deckhouse.io/application-instance` label with the instance name is required. The web interface looks for credentials only among the Secrets with this label in the namespace of the application. Deckhouse Platform doesn’t add this label to the objects, and the `packages.deckhouse.io/instance` label that Deckhouse Platform adds (for details, refer to “Labels and object protection”) is not taken into account.
- The `packages.deckhouse.io/application-ingress` label binds the Secret to an Ingress from the templates by its name. For an application published through the `alb` module, use the `packages.deckhouse.io/application-alb` label with the name of an HTTPRoute from the templates instead, and for a ServiceAccount token, the `packages.deckhouse.io/application-serviceaccount` label with the name of a ServiceAccount from the templates. The web interface shows the credentials next to the endpoints with the description of this object. A label value can’t be longer than 63 characters, so the name of the Ingress, HTTPRoute, or ServiceAccount must fit this limit.
- Instead of the label, you can set the endpoint description in the `packages.deckhouse.io/application-endpoint-description` annotation of the Secret. If both are set, the description of the Ingress, HTTPRoute, or ServiceAccount is used.
- The web interface shows the credentials only if the application has endpoints with a matching description. Endpoints without a description don’t get credentials, so set a description in the annotation of the Ingress, HTTPRoute, or ServiceAccount instead of `"true"`.
- The `packages.deckhouse.io/initial-secret` label marks the Secret as one-time credentials. Deckhouse Platform creates such a Secret once and doesn’t update it afterwards: it adds the `werf.io/resource-policy: skip-update` annotation to the Secret (an existing policy is kept, for example, `skip-delete` becomes `skip-delete,skip-update`). The admission policies let users change such a Secret, so the web interface can mark it as shown, and users can change the credentials. Without this label, the policies deny the change, and the credentials can be requested

again. Deckhouse Platform checks only that the label is present, its value doesn’t matter.

Put the login into the `login` key and the password into the `password` key. The web interface shows every key of `data` as a separate field named after the key, so don’t store other data in this Secret.

In the web interface, endpoints with credentials get a button for getting the login and password. The credentials can be obtained only once: after showing them, the web interface adds the `packages.deckhouse.io/showed-credentials: "true"` annotation to the Secret and doesn’t offer this Secret again. This restriction applies only to the web interface: users with permission to read Secrets in the namespace can still read the Secret.

Since Deckhouse Platform doesn’t update a Secret with the `packages.deckhouse.io/initial-secret` label, changes to its template, including `data`, apply only when the Secret is created, for example, in a new instance.

Generate the credentials once and keep them unchanged: the web interface doesn’t offer a Secret again even if its data changes. Don’t generate the password in the template: functions such as `randAlphaNum` give a new value on every render (for details, refer to “Known issues and troubleshooting”). Instead, for example, generate it in a hook if the Secret doesn’t exist yet, and otherwise take it from the snapshot of the existing Secret. Values set by hooks are kept only in memory, so the hook must restore the password after Deckhouse Platform restarts (for details, refer to “Hooks”).

ServiceAccount token

To show a ServiceAccount token, add a Secret of the `kubernetes.io/service-account-token` type for the ServiceAccount from the example above:

```
apiVersion: v1
kind: Secret
metadata:
  name: d8a-{{ .Application.Instance.Name }}-api-token
  labels:
    packages.deckhouse.io/application-instance: {{ .Application.Instance.Name }}
    packages.deckhouse.io/application-serviceaccount: d8a-{{ .Application.Instance.Name
}}-api
    packages.deckhouse.io/initial-secret: "true"
  annotations:
    kubernetes.io/service-account.name: d8a-{{ .Application.Instance.Name }}-api
type: kubernetes.io/service-account-token
```

Kubernetes fills in the `ca.crt`, `namespace`, and `token` keys of this Secret: the CA certificate of the Kubernetes API, the namespace, and the ServiceAccount token. The web interface shows them as separate fields.

Restrictions

Templates must follow the Application constraints:

- Create only namespaced objects. Don’t set `metadata.namespace`: DP creates the objects in the namespace of the Application, and `d8 package verify` reports an error for objects with this field (the `instance-namespace` rule).
- Don’t create CustomResourceDefinition objects. DP doesn’t install CRDs from the `crds/` directory of the chart.

By default, `d8 package verify` also requires that:

- Every Deployment and StatefulSet has a PodDisruptionBudget whose selector matches the labels of its Pods (the `pdb` rule).
- Every Deployment, StatefulSet, and DaemonSet has a VerticalPodAutoscaler with a policy for each container (the `vpa` rule). The VerticalPodAutoscaler kind is available only if the `vertical-pod-autoscaler` module is enabled, so render it under the `.Capabilities.APIVersions.Has "autoscaling.k8s.io/v1/VerticalPodAutoscaler"` condition, as the package skeleton does.
- Services refer to container ports by name in `targetPort` (the `service-port` rule).

You can lower the severity of these rules in `.pkglint.yaml` (for details, refer to “Linting”).

Replica limits

The user can change the number of replicas of a Deployment, StatefulSet, or ReplicaSet in the `spec.resourceRequests` field of the Application. If the application can’t run with an arbitrary number of replicas, add one of the following annotations to the workload in the templates:

- `packages.deckhouse.io/scale-disabled: "true"`: The number of replicas can’t be changed.
- `packages.deckhouse.io/scale-down-disabled: "true"`: The number of replicas can’t be lower

than the number set in the templates.

The limits are counted from `spec.replicas` of the workload rendered from the templates (`1` if the field isn't set), not from the current number of replicas. For example, if the templates set `replicas: 2` and `scale-down-disabled`, the user can scale the workload to 4 replicas and back to 2, but not to 1. Only the `"true"` value enables a limit.

If a HorizontalPodAutoscaler from the templates manages the replicas of the workload, add the annotation to the HorizontalPodAutoscaler. Deckhouse Platform sets both `minReplicas` and `maxReplicas` of such a HorizontalPodAutoscaler to the requested number of replicas, and the annotations limit this number:

- `scale-disabled`: The number must be equal to both `minReplicas` and `maxReplicas`. If they differ, the number of replicas can't be changed.
- `scale-down-disabled`: The number can't be lower than `minReplicas` (`1` if the field isn't set).

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: d8a-{{ .Application.Instance.Name }}-web
  annotations:
    packages.deckhouse.io/scale-down-disabled: "true"
spec:
  replicas: 2
  # ...
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: d8a-{{ .Application.Instance.Name }}-api
  annotations:
    packages.deckhouse.io/scale-down-disabled: "true"
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: d8a-{{ .Application.Instance.Name }}-api
  minReplicas: 2
  maxReplicas: 10
  # ...
```

If `spec.resourceRequests` requests a forbidden number of replicas, Deckhouse Platform doesn't apply the templates, and the objects of the application stay as they were. The Application itself is accepted, because Deckhouse Platform checks the limits when it renders the templates. The conditions of the Application get the `ScalingForbidden` reason, and the condition message names the workload, for example: `Deployment 'd8a-myapp-web' does not allow its replicas to go below 2`.

Workload health

DP determines whether the application is running from its Deployments and StatefulSets only:

- The `Scaled` condition of the Application becomes `True` when all Deployments and StatefulSets rendered by the templates have rolled out and have the desired number of ready replicas. DaemonSets, Jobs, CronJobs, and Pods are not taken into account.
- A Deployment whose rollout exceeds `spec.progressDeadlineSeconds` sets `Scaled` to `False` with the `Degraded` reason.
- The first installation is considered complete (`Installed=True`) only after the manifests are applied and `Scaled` becomes `True`.

⚠ The package must contain at least one Deployment or StatefulSet. Otherwise, the `Scaled` condition stays `Unknown`, and the application never reaches the `Installed` and `Ready` states.

Additional resources

- Global parameters
- Global custom resources
- Module `alb` documentation
- Module `cert-manager` documentation

- Module `vertical-pod-autoscaler` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Nelm annotations

Application templates are rendered and deployed by Nelm. Nelm extends the standard Helm behavior with annotations that control deployment order, resource lifecycle, readiness tracking, and log output. This page covers the most commonly used annotations.

Deployment stages

Nelm processes a deployment in three stages. Different annotation groups affect different stages.

Render

Templates are evaluated with the current values. Deckhouse Platform (DP) renders the templates with access to the cluster: `.Capabilities` reflects the API server, and `lookup` returns live objects. At this stage:

- Template functions (`include`, `tpl`, `lookup`, and others) are executed.
- All resources are rendered and stored in the release, regardless of `werf.io/deploy-on`.

Plan

Nelm connects to the cluster, reads current resource state, and runs a **dry-run Server-Side Apply** to compute the exact diff. It then builds a **DAG of operations** — which resources to create, update, or delete, and in what order.

At this stage:

- `werf.io/deploy-on` determines for which operations (`install`, `upgrade`, and so on) and in which stage the resource is deployed.
- Order and dependency annotations (`werf.io/weight`, `werf.io/deploy-dependency-*`, `*.external-dependency.werf.io/*`) shape the DAG.
- Lifecycle annotations (`werf.io/ownership`, `werf.io/delete-policy`, `werf.io/delete-propagation`) determine which operations enter the plan.

Dry-run SSA means the diff is computed by the API server — defaulting, admission plugins, and mutating webhooks are all applied.

Apply

The DAG is executed: resources are created, updated, or deleted in dependency order, with parallelism where no dependencies exist. Readiness tracking and log streaming run concurrently.

At this stage:

- Tracking annotations (`werf.io/track-termination-mode`, `werf.io/fail-mode`, `werf.io/failures-allowed-per-replica`, `werf.io/no-activity-timeout`) control waiting behavior.
- Log annotations control what is printed during deployment.

Common use cases

Job that must run before the main application

Database migration before the application starts:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: d8a-{{ .Application.Instance.Name }}-db-migrate
  annotations:
    werf.io/delete-policy: before-creation
spec: ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: d8a-{{ .Application.Instance.Name }}-app
  annotations:
    werf.io/deploy-dependency-migrate: state=ready,kind=Job,name=d8a-{{
.Application.Instance.Name }}-db-migrate
spec: ...
```

The annotation `werf.io/delete-policy: before-creation` makes Nelm recreate the Job on every deployment, so the migration runs each time. By default, Nelm recreates a Job only when an immutable field of the Job changes. The Deployment waits for `ready`, that is, for the Job to complete successfully.

Preserve a resource across uninstall or chart removal

A PVC with database data that must not be deleted even when the Application is uninstalled:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: d8a-{{ .Application.Instance.Name }}-postgres-data
  annotations:
    helm.sh/resource-policy: keep
spec:
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 50Gi
  storageClassName: gp3
```

The annotation `helm.sh/resource-policy: keep` prevents the PVC from being deleted on uninstall or when removed from the chart. It doesn’t protect against `d8 k delete pvc`. To keep the data in this case, use a StorageClass with `reclaimPolicy: Retain`: then the PersistentVolume outlives the PVC.

Resource shared between releases

A TLS Secret used by multiple charts should not disappear when any one of them is uninstalled:

```
apiVersion: v1
kind: Secret
metadata:
  name: d8a-{{ .Application.Instance.Name }}-shared-tls
  annotations:
    werf.io/ownership: anyone
type: kubernetes.io/tls
data: ...
```

The annotation `werf.io/ownership: anyone` tells Nelm it is not the sole owner, so it skips deletion on uninstall. DP still deletes the Secret together with the Application if its kind is listed in `orphanResources`. To keep the Secret, add `helm.sh/resource-policy: keep`.

Wait for a resource created by an operator

Deploy only after `cert-manager` creates the Certificate object:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: d8a-{{ .Application.Instance.Name }}-app
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/d8a-{{
.Application.Instance.Name }}-tls
spec: ...
```

Nelm waits until the Certificate object exists before creating the Deployment. Nelm doesn’t wait until the certificate is issued.

Non-critical component that must not fail the deploy

A metrics DaemonSet whose unavailability should not block the release:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: d8a-{{ .Application.Instance.Name }}-metrics-agent
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
spec: ...
```

Nelm will not wait for readiness and will not fail on timeout.

Resource rendered only on first install

An init Job that runs only during `install`, not `upgrade`:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: d8a-{{ .Application.Instance.Name }}-init-data
  annotations:
    werf.io/deploy-on: install
    werf.io/ownership: anyone   # CRITICAL
spec: ...
```

Without `werf.io/ownership: anyone`, on upgrade the resource renders as absent and the owning release removes it. `anyone` prevents that. Nelm doesn’t delete such a Job on uninstall either. To make DP delete it together with the Application, list `Job` in `orphanResources`.

Slow-starting resource with a large image

StatefulSet with a large Docker image (ML models, Elasticsearch with pre-loaded indices):

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: d8a-{{ .Application.Instance.Name }}-elasticsearch
  annotations:
    werf.io/no-activity-timeout: 20m
    werf.io/failures-allowed-per-replica: "3"
    werf.io/show-service-messages: "true"
spec: ...
```

Used annotations:

- `no-activity-timeout: 20m`: 20 minutes without events before Nelm considers it timed out. Use this for slow image pulls or long initialization.
- `failures-allowed-per-replica: "3"`: Allows up to 3 tracking errors per replica, for example, container restarts or image pull errors, before the deployment fails. Use for flapping init dependencies.
- `show-service-messages: "true"`: Emit Kubernetes Events in deploy output (useful for diagnosing `ImagePullBackOff`, scheduling failures, OOM events).

Ordering and dependencies

Weights

The annotation `werf.io/weight` groups resources: equal weights deploy in parallel; different weights deploy sequentially in ascending order. The weight is ignored for a resource with a `werf.io/deploy-dependency-*` annotation.

```
metadata:
  annotations:
    # Deploys before default-weight (0) resources.
    werf.io/weight: "-10"
```

Direct dependencies

The annotation `werf.io/deploy-dependency-<id>` waits for a specific resource in the same release to reach a specific state before deploying the annotated resource.

```
metadata:
  annotations:
    werf.io/deploy-dependency-db: state=ready,kind=StatefulSet,name=d8a-{{
.Application.Instance.Name }}-postgres
    werf.io/deploy-dependency-migrations: state=present,kind=Job,name=d8a-{{
.Application.Instance.Name }}-db-migrate
```

Dependency states:

- `ready`: Resource is in a ready state (for example, `availableReplicas == replicas` for a Deployment).
- `present`: Resource exists in the cluster.

Full format:

```
werf.io/deploy-dependency-<id>: state=ready|present[,name=<name>][,namespace=<namespace>][,kind=<kind>]
[,group=<group>][,version=<version>]
```

⚠ This annotation has no effect if the dependency resource is in a different deploy stage (pre/main/post). Cross-stage ordering is already enforced by the stage sequence itself.

External dependencies

The annotation `<id>.external-dependency.werf.io/resource` waits until a resource **outside the release** (created by an operator or another release) exists. Nelm doesn’t wait for the readiness of the resource:

```
metadata:
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/d8a-{{
.Application.Instance.Name }}-tls
    cert.external-dependency.werf.io/namespace: myapp-production  # Namespace of the
external resource
```

Full format:

```
<id>.external-dependency.werf.io/resource: <kind>[.<version>.<group>]/<name>
```

Delete dependencies

The annotation `werf.io/delete-dependency-<id>` means that the annotated resource will be deleted only after the referenced resource becomes `absent`:

```
metadata:
  annotations:
    werf.io/delete-dependency-app: state=absent,kind=Deployment,name=d8a-{{
.Application.Instance.Name }}-app
```

Lifecycle annotations

helm.sh/resource-policy

The `keep` value instructs Nelm not to delete the resource on uninstall or when removed from the chart. The resource continues to be updated on install/upgrade as long as it renders. If `werf.io/resource-policy` is set, Nelm ignores `helm.sh/resource-policy`. DP doesn't delete a resource with `helm.sh/resource-policy: keep` as an orphan resource, even if `werf.io/resource-policy` is set: only this annotation protects a resource from such deletion.

werf.io/resource-policy

A list of policies:

- `skip-create`: Do not create the resource if it doesn't exist in the cluster.
- `skip-update`: Do not update the resource if it already exists.
- `skip-recreate`: Do not recreate the resource when a recreation would otherwise be required.
- `skip-delete` (alias: `keep`): Do not delete the resource.

DP deletes a resource with `skip-delete` or `keep` together with the Application if its kind is listed in `orphanResources`. To keep the resource, also set `helm.sh/resource-policy: keep`.

werf.io/ownership

Allowed values:

- `release` (default for regular resources): Resource is deleted on uninstall and when absent from the chart. Release metadata annotations are applied.
- `anyone` (default for hooks): Resource is not deleted on uninstall. Release metadata annotations are not applied.

Use `anyone` for resources shared between releases or for resources that should survive their release (init Jobs with `werf.io/deploy-on: install`).

Resources that stay after uninstall, that is, hooks and resources with `werf.io/ownership: anyone`, are deleted by DP together with the Application if their kind is listed in `orphanResources`. To keep such a resource, set `helm.sh/resource-policy: keep`.

werf.io/deploy-on

Controls on which lifecycle operations the resource is rendered.

```
werf.io/deploy-on: pre-install,upgrade,post-install
```

Allowed values:

- `pre-install`
- `install`
- `post-install`
- `pre-upgrade`
- `upgrade`
- `post-upgrade`
- `pre-rollback`
- `rollback`
- `post-rollback`
- `pre-delete`
- `delete`
- `post-delete`

Default for regular resources: `install,upgrade,rollback`. Hooks are deployed on the events listed in `helm.sh/hook`.

> ⚠ If a resource is rendered for `install` but not `upgrade`, and `werf.io/ownership: release` is set, the resource will be **deleted on upgrade** because it is absent from the upgrade render. Set `werf.io/ownership: anyone` to prevent this.

werf.io/delete-policy

Controls when a resource is deleted relative to the `apply` operation.

Value	When
`before-creation`	Always recreate before `apply` (default for hooks)
`before-creation-if-immutable`	Recreate only on `field is immutable` error (default for Jobs)
`succeeded`	Delete after successful deploy
`failed`	Delete on readiness-check failure

Values can be combined with a comma: `before-creation,succeeded`.

werf.io/delete-propagation

Kubernetes deletion propagation strategy.

Value	Behavior
`Foreground` (default)	Wait for dependent objects to be deleted
`Background`	Delete resource immediately; dependents deleted asynchronously
`Orphan`	Delete resource; leave dependents

Tracking annotations

Annotation	Default	Description
`werf.io/track-termination-mode`	`WaitUntilResourceReady`	`WaitUntilResourceReady` or `NonBlocking`
`werf.io/fail-mode`	`FailWholeDeployProcessImmediately`	`FailWholeDeployProcessImmediately` or `IgnoreAndContinueDeployProcess`
`werf.io/failures-allowed-per-replica`	`1` for Deployment, StatefulSet, and DaemonSet, `0` for other kinds	Number of tracking errors per replica, for example, container restarts or image pull errors, allowed before the deployment fails. Ignored for Jobs
`werf.io/no-activity-timeout`	`4m`	Go duration; timeout on absence of events or status changes
`werf.io/show-service-messages`	`false`	Show Kubernetes Events in deploy output

Log annotations

> **i** DP doesn’t collect pod logs when it deploys Applications, so these annotations don’t affect the deployment.

Annotation	Default	Description
`werf.io/skip-logs`	`false`	Suppress all pod logs
`werf.io/skip-logs-for-containers`	—	Comma-separated list of containers to suppress
`werf.io/show-logs-only-for-containers`	—	Comma-separated list; only show logs for these containers
`werf.io/show-logs-only-for-number-of-replicas`	`1`	Show logs only for the first N replicas
`werf.io/log-regex`	—	RE2 pattern; show only matching lines

Annotation	Default	Description
`werf.io/log-regex-skip`	—	RE2 pattern; suppress matching lines
`werf.io/log-regex-for-<container>`	—	Per-container RE2 filter for showing
`werf.io/log-regex-skip-for-<container>`	—	Per-container RE2 filter for suppressing

Template functions

werf_secret_file

Embed the decrypted content of a file from the `secret/` directory:

```
data:
  config.yaml: {{ werf_secret_file "config.yaml" | b64enc }}
```

⚠ DP doesn’t provide a decryption key for Nelm secrets, so Application packages can’t use `werf_secret_file`, the `secret/` directory, or `secret-values.yaml`.

dump_debug, printf_debug, include_debug, tpl_debug

The functions `dump_debug` and `printf_debug` render nothing, and `include_debug` and `tpl_debug` render the same output as `include` and `tpl`. The functions write to the log only when Nelm debug logging is enabled. DP doesn’t enable it, so in Application packages these functions log nothing.

```
{{ dump_debug $ }}
{{ printf_debug "replicaCount: %d" .Values.replicaCount }}
{{ include_debug "myapp.labels" . | nindent 4 }}
{{ tpl_debug "{{ .Values.template }}" . }}
```

Known issues and troubleshooting

- **`null` values and SSA.** Server-Side Apply frequently fails on fields with a `null` value. If `.Values.foo` is `nil`, the manifest gets `foo: null`. Fix with `{{ if .Values.foo }}` guards or `default`.
- **`lookup` timing.** If cluster resources change between the Plan and Apply stages, the rendered plan may be stale. Avoid `lookup` for critical logic; pass data through values instead.
- **Non-deterministic functions.** `now`, `randAlphaNum`, `uuidv4`, and the unsorted output of `keys` produce different results on every render. DP applies the templates when the rendered manifests change, so such functions cause a redeployment in every reconciliation run.
- **`werf.io/deploy-dependency-*` and stages.** The annotation has no effect across deploy stages (pre/main/post). Cross-stage ordering is implicit in the stage sequence.
- **Resources that disappear after deployment.** DP checks every 4–5 minutes that all regular (non-hook) resources of the release exist and applies the templates again if one is missing (for details, refer to “Resource restoration”). Therefore, a regular Job that is deleted after it completes, for example, because of `werf.io/delete-policy: succeeded` or `ttlSecondsAfterFinished`, runs again after every check, and a deleted resource that is rendered only on install causes a redeployment after every check. Make such Jobs Helm hooks: DP doesn’t check hooks.

Additional resources

- Module `cert-manager` documentation

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hooks

Application hooks are written in Go using module-sdk. Deckhouse Platform (DP) runs them at certain stages of the application lifecycle, on changes of Kubernetes objects, and on schedule. Hooks prepare values for templates, create and change objects in the application namespace, and validate settings.

Hooks binary

All hooks of a package are compiled into a single Go binary. In the package skeleton, the sources are located in `hooks/batch/`, and `d8 package build` builds the binary according to `hooks/hooks.yaml` and puts it into the package bundle.

Each hook is registered with `registry.RegisterFunc`, and the `main` function calls `app.Run`:

```
// hooks/batch/main.go
package main

import (
    "github.com/deckhouse/module-sdk/pkg/app"

    "hooks/settings"
    _ "hooks/triggers"
)

func main() {
    app.Run(app.WithSettingsCheck(settings.Check))
}
```

Requirements for the binary:

- The hook source files must be located under a directory named `hooks`: module-sdk builds the hook name from the file path.
- The binary must register at least one hook. DP ignores a binary without hooks, even if it contains a settings validation hook.
- Build a static binary for Linux (`CGO_ENABLED=0`), as the skeleton does.

DP finds hooks by running each executable file without an extension from the `hooks/` directory of the bundle with the `hook list` argument. A binary that fails at this stage, for example, because of an invalid hook configuration, is skipped without an error in the Application status. To make sure that the hooks are registered, check the output of the following commands in CI:

```
cd hooks/batch
CGO_ENABLED=0 go build -o /tmp/hooks .
/tmp/hooks hook list
/tmp/hooks hook config
```

Hook example

```
package state

import (
   "context"

   corev1 "k8s.io/api/core/v1"
   metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"

   "github.com/deckhouse/module-sdk/pkg"
   objectpatch "github.com/deckhouse/module-sdk/pkg/object-patch"
   "github.com/deckhouse/module-sdk/pkg/registry"
)

var _ = registry.RegisterFunc(&pkg.ApplicationHookConfig{
   OnBeforeHelm: &pkg.OrderedConfig{Order: 10},
   Kubernetes: []pkg.ApplicationKubernetesConfig{
      {
         Name:      "credentials",
         APIVersion: "v1",
         Kind:      "Secret",
         LabelSelector: &metav1.LabelSelector{
            MatchLabels: map[string]string{"example.com/credentials": "true"},
         },
         JqFilter: `{"name": .metadata.name}`,
      },
   },
}, handle)

type secret struct {
   Name string `json:"name"`
}

func handle(_ context.Context, input *pkg.ApplicationHookInput) error {
   secrets, err := objectpatch.UnmarshalToStruct[secret](input.Snapshots, "credentials")
   if err != nil {
      return err
   }

   // The path must be declared in openapi/values.yaml.
   input.Values.Set("internal.credentialsCount", len(secrets))

   // The ConfigMap is created as d8a-<INSTANCE_NAME>-state in the application
namespace.
   input.PatchCollector.CreateOrUpdate(&corev1.ConfigMap{
      TypeMeta:   metav1.TypeMeta{APIVersion: "v1", Kind: "ConfigMap"},
      ObjectMeta: metav1.ObjectMeta{Name: "state"},
      Data:      map[string]string{"instance": input.Instance.Name()},
   })

   return nil
}
```

Bindings and execution order

The `pkg.ApplicationHookConfig` structure defines when the hook runs:

Field	When the hook runs
`OnStartup`	Once after the package is loaded: on installation, after the package version changes, and after DP restarts. The hook doesn’t get snapshots
`OnBeforeHelm`	In every reconciliation run, before the templates are applied
`OnAfterHelm`	In every reconciliation run, after the templates are applied. If the hook changes values, DP applies the templates again

Field	When the hook runs
`OnBeforeDeleteHelm`	Before the Helm release is uninstalled: when the Application is deleted or when the package requirements stop being met. The hook doesn't run when the package version changes
`OnAfterDeleteHelm`	After the Helm release is uninstalled, in the same cases as `OnBeforeDeleteHelm`
`Schedule`	On schedule in the crontab format
`Kubernetes`	When the watched objects in the application namespace change, and during the synchronization of the binding (for details, refer to "Kubernetes bindings")

Other fields:

- `Queue`: Name of the queue for the `Schedule` and `Kubernetes` bindings of the hook (default: `main`). These bindings run in a queue separate from the lifecycle stages, so they can run at the same time as `OnBeforeHelm` or `OnAfterHelm` hooks.
- `AllowFailure` and `Settings` are not supported for applications.

Hooks with the same binding run in the ascending order of `Order` (the `pkg.OrderedConfig` field). The sequence of lifecycle stages is described in "Lifecycle and debugging".

A hook can't combine `OnStartup` with `Kubernetes` bindings: module-sdk panics when such a hook is registered, and DP skips the whole binary.

Error handling

- If a hook fails, the stage that ran it fails and is retried with an increasing delay, from 15 seconds up to 2 hours. The next tasks of the application wait. The error is reported in the Application's `status.conditions` and `status.summary` fields.
- If an `OnBeforeDeleteHelm` hook fails, DP doesn't uninstall the Helm release and retries the hook. The Application is not deleted until the hook succeeds.
- If an object operation of the hook fails, the hook run fails, and the values changes made by the run are discarded. The operations that have already been executed are not rolled back.

> ⚠ If a hook handler panics, module-sdk recovers from the panic and considers the hook execution successful. However, the handler results are not applied: changes to values and object operations are ignored. Do not use panic in hook handlers. Return errors instead.

ApplicationHookInput

The hook handler has the `func(ctx context.Context, input *pkg.ApplicationHookInput) error` signature. The `pkg.ApplicationHookInput` structure contains the following fields:

Field	Description
`Instance`	Instance parameters: `Name()` and `Namespace()`
`Snapshots`	Snapshots of the Kubernetes bindings of the hook
`Values`	Application values: reading (`Get`, `GetOk`, `Exists`, `ArrayCount`) and changing (`Set`, `Remove`)
`Settings`	Effective settings of the application, read-only
`PatchCollector`	Operations with objects in the application namespace (for details, refer to "Object operations")
`DC`	Dependencies: an HTTP client (`GetHTTPClient`), a container registry client (`GetRegistryClient`), and a clock (`GetClock`)
`Logger`	Logger. The output is written to the DP log
`MetricsCollector`	Not supported for applications: DP ignores the metrics

Application hooks don't get a Kubernetes client. To read objects, use Kubernetes bindings, and to change them, use `PatchCollector`.

The instance name and namespace are the only parameters of the instance available to hooks. The package name, version, and images are available only to templates.

Kubernetes bindings

The `pkg.ApplicationKubernetesConfig` structure describes the objects the hook watches:

Field	Description
`Name`	Binding name, the key in `Snapshots`
`APIVersion`, `Kind`	API version and kind of the objects. Only namespaced kinds are supported
`NameSelector`, `LabelSelector`, `FieldSelector`	Selection of objects
`JqFilter`	jq expression applied to each object. The snapshot contains the results of the expression. Without the expression, the snapshot contains full objects
`ExecuteHookOnEvents`	`false` — update the snapshot without running the hook when the objects change. Default: `true`
`ExecuteHookOnSynchronization`	`false` — don’t run the hook during the synchronization. Default: `true`
`WaitForSynchronization`	`false` — don’t wait for the synchronization of the binding before the next stages. It takes effect only if the `Queue` field of the hook is set. Default: `true`
`AllowFailure`	Ignore errors of the hook during the synchronization

Behavior:

- DP always restricts the bindings to the application namespace, so the structure has no namespace selector.
- On every run, the hook gets the snapshots of all its bindings: the current state of the objects, not the event that triggered the run.
- `OnBeforeHelm`, `OnAfterHelm`, `OnBeforeDeleteHelm`, and `OnAfterDeleteHelm` hooks get the snapshots too.
- DP synchronizes the bindings at the beginning of every reconciliation run: hooks with `ExecuteHookOnSynchronization` enabled run with the full list of objects.
- `NameSelector` matches the actual object names, including the `d8a-<INSTANCE_NAME>-` prefix.

The hook configuration is the same for all instances of the package, so selectors can’t refer to the instance name. If several instances of the package can be installed into the same namespace, add the instance label to the snapshot and compare it with `input.Instance.Name()`. Objects rendered from the templates carry the `packages.deckhouse.io/instance` label:

```
JqFilter: `{"name": .metadata.name, "instance":
.metadata.labels["packages.deckhouse.io/instance"]}`,
```

Object operations

`input.PatchCollector` collects the operations with objects. DP executes them after the hook completes:

Method	Description
`Create`	Create an object. The operation fails if the object exists
`CreateIfNotExists`	Create an object if it doesn’t exist
`CreateOrUpdate`	Create an object or replace the existing one. The fields that the hook doesn’t set are removed from the object
`Delete`, `DeleteInBackground`, `DeleteNonCascading`	Delete an object with foreground, background, or orphan propagation of the deletion. A missing object is ignored
`PatchWithJSON`, `PatchWithMerge`, `PatchWithJQ`	Patch an object with a JSON Patch (RFC 6902), a JSON Merge Patch (RFC 7396), or a jq expression. To patch a subresource, pass the `objectpatch.WithSubresource("status")` option

Behavior:

- module-sdk sets the application namespace in every operation, regardless of the namespace specified in the object.
- Typed objects must have `TypeMeta` with `apiVersion` and `kind` set. Otherwise, the operation is not executed.
- Patching a missing object fails the hook. Check that the object exists, for example, in a snapshot, before patching it.
- Objects created by hooks are not part of the Helm release, and DP doesn't add platform labels to them, so uninstalling the release doesn't delete them. To delete such objects together with the application, set the `packages.deckhouse.io/instance` label to the instance name and list their kind in `orphanResources`. You can also set an owner reference to an object of the release, or delete the objects in `OnBeforeDeleteHelm` or `OnAfterDeleteHelm` hooks.

Object names

DP adds the `d8a-<INSTANCE_NAME>-` prefix to the object name in every operation, unless the name already starts with it. For example, for the `myapp` instance:

- `Create` of the `state` ConfigMap creates `d8a-myapp-state`.
- `PatchWithMerge` of the `server` Deployment changes `d8a-myapp-server`, that is, the Deployment named `d8a-<INSTANCE_NAME>-server` in the templates.

As a result, a hook can change only the objects of its own instance. Objects with the `d8a-` prefix are protected from changes by users (for details, refer to "Templates").

Values

`input.Values` contains the application values: the same document the templates get as `.Values`. The settings are located at the root of the document, without the package name key.

- `Set(path, value)` and `Remove(path)` accept a path with dot-separated keys, for example, `internal.credentialsCount`. The parent object of the path must exist, so declare intermediate objects in `openapi/values.yaml` with `default: {}`.
- After the hook completes, DP validates the values against `openapi/values.yaml`. The schema must include the settings schema with `x-extend` (for details, refer to "Application settings"): the settings are located at the root of the values, and undeclared fields fail the validation.
- The values changes are kept in memory. After DP restarts or the package version changes, they are lost until the hooks run again.

What happens after the values change depends on the binding:

Binding	Result
`Kubernetes`, `Schedule`	DP starts a new reconciliation run
`OnAfterHelm`	DP applies the templates again
`OnStartup`, `OnBeforeHelm`, synchronization	The values are used when the templates are applied in the same run

`input.Settings` is read-only: hooks can't change the application settings.

Settings validation hook

Applications can include a settings validation hook for checks that the OpenAPI schema can't express, for example, business logic constraints across multiple settings fields.

The hook implements a `Check` function with the signature:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result
```

`settingscheck.Input` provides access to the settings (`input.Settings`) with the default values from `openapi/settings.yaml` applied, as well as to a logger (`input.Logger`).

`settingscheck.Result` is one of:

- `settingscheck.Allow(warnings...)` — settings are valid; optionally attach warning messages.
- `settingscheck.Reject(reason)` — settings are invalid.

Example:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result {
   replicas := input.Settings.Get("replicas").Int()
   if replicas == 0 {
      return settingscheck.Reject("replicas cannot be 0")
   }

   var warnings []string
   if replicas == 2 {
      warnings = append(warnings, "an even number of replicas may cause split-brain in
some configurations")
   }

   if replicas > 3 {
      return settingscheck.Reject("replicas cannot be greater than 3")
   }

   return settingscheck.Allow(warnings...)
}
```

To register the check, pass it to `app.Run` in the hooks binary: `app.Run(app.WithSettingsCheck(Check))`. A package can contain only one settings validation hook.

DP calls the check in the following cases:

- When the settings of an installed application are changed, and `spec.packageVersion` matches the installed version, the check runs in the validating webhook. A rejection denies the request with the specified reason. Warnings are returned to the client, for example, `d8 k` prints them.
- When DP applies changed settings. A rejection leaves the new settings unapplied, and the error is reported in the Application conditions. Warnings are ignored.

The check doesn't run in the validating webhook when an Application is created or `spec.packageVersion` changes: in these cases, the request is checked against the settings schema only. Describe critical constraints in the schema, for example, with CEL rules, to reject invalid settings before installation.

Restrictions

- Kubernetes bindings and object operations are limited to the application namespace, and object operations are also limited to the objects with the `d8a-<INSTANCE_NAME>-` name prefix.
- Hooks can't create or change cluster-wide objects.
- Readiness probes (`app.WithReadiness`) are intended for modules. Don't use them in applications: DP determines the application health from its workloads.
- Metrics of hooks are not collected.

Additional resources

- Global custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Lifecycle and debugging

This page describes how Deckhouse Platform (DP) runs an application instance and which tools help to debug a package.

Admission checks

When an Application is created or changed, the DP validating webhook checks that:

- The Application name is at most 24 characters long.
- The ApplicationPackage exists, and the ApplicationPackageVersion for the specified repository, package, and version exists and has its metadata loaded.
- The settings match the settings schema of the package version (for details, refer to "Application settings").
- The cluster meets the package requirements: the Kubernetes and DP versions and the module dependencies. The rejection message names the unmet requirement, the required value, and the value

in the cluster.

Installation pipeline

DP processes each application instance in a separate queue, step by step:

1. **Download.** DP downloads the package bundle of the specified version from the repository.
2. **Load.** DP reads `package.yaml`, the schemas, and `images_digests.json`, and discovers the hooks.
3. **Scheduling.** DP waits until the package requirements are met: the cluster bootstrap is complete, the Kubernetes and DP versions match the constraints, and the modules from `requirements.modules` are enabled. While DP waits for the modules the application depends on, the Application's `status.summary` field reports it.
4. **Configuration.** DP validates and applies the settings.
5. **Hook initialization.** DP enables the `schedule` bindings and the Kubernetes monitors of the hooks and runs the `Synchronization` of the Kubernetes bindings. The first time after the package is loaded, DP also runs `onStartup` hooks.
6. **Apply.** DP runs `beforeHelm` hooks, renders the templates and applies them with Nelm, and then runs `afterHelm` hooks. If `afterHelm` hooks change values, DP applies the templates again.

Steps 4–6 form a reconciliation run. In the apply step, DP upgrades the Helm release only if it is needed: on the first installation, if the last release revision is not deployed, if the rendered manifests have changed, or if some resources are missing in the cluster.

The order of hook bindings and the handling of hook errors are described in "Hooks".

Events and reactions

Event	Reaction of DP
The `spec.settings` or `spec.maintenance` field of the Application changes	A new reconciliation run
A `schedule` or Kubernetes hook changes values	A new reconciliation run
The `spec.packageVersion` field changes	DP stops the hooks of the current version while keeping its resources, and then goes through all steps with the new version. The templates of the new version are applied as an upgrade of the same Helm release
DP restarts	DP loads the package again and goes through all steps, including `onStartup` hooks
A resource of the application is deleted from the cluster	DP restores it (for details, refer to "Resource restoration")
The package requirements stop being met, for example, a module from `requirements.modules.mandatory` is disabled	DP uninstalls the application but keeps the Application (for details, refer to "Suspension")
The Application is deleted	DP uninstalls the application, deletes its orphan resources, and then removes the Application (for details, refer to "Deletion")

Suspension

DP checks the package requirements not only on admission but all the time. If the requirements of an installed application stop being met, DP does the following:

1. Runs `beforeDeleteHelm` hooks.
2. Uninstalls the Helm release, which deletes the resources of the application.
3. Runs `afterDeleteHelm` hooks.

The Application stays in the cluster in the `Suspended` state, and the orphan resources of the application are not deleted. When the requirements are met again, DP installs the application again with the same settings.

⚠ The resources are deleted together with the Helm release, including PersistentVolumeClaims. To keep the data, add the `helm.sh/resource-policy: keep` annotation to PersistentVolumeClaims (for details, refer to "Nelm annotations").

Deletion

When an Application is deleted, DP does the following:

1. Runs `beforeDeleteHelm` hooks.
2. Uninstalls the Helm release and waits until its resources are deleted.
3. Runs `afterDeleteHelm` hooks and stops the hooks: disables the `schedule` bindings and the Kubernetes monitors.
4. Deletes the orphan resources of the application.
5. Removes the Application.

If a step fails, DP retries it with an increasing delay, from 15 seconds up to 2 hours. The Application stays in the cluster until all steps succeed.

Orphan resources

Uninstalling the Helm release deletes only the objects of the release. The following objects stay in the namespace:

- Objects that the workloads or hooks of the application create at runtime.
- Objects of the templates that Nelm doesn't delete when it uninstalls a release, if they are still in the cluster (for details, refer to "Nelm annotations"):
  - Helm hooks, that is, objects with the `helm.sh/hook` annotation.
  - Objects with the `werf.io/ownership: anyone` annotation, for example, a Job that is deployed only on install.
  - Objects with the `helm.sh/resource-policy: keep` annotation and objects whose `werf.io/resource-policy` annotation contains `keep` or `skip-delete`.

To make DP delete such objects together with the Application, list their kinds in the `orphanResources` field of the values, as the default value of the field in `openapi/values.yaml`. For example:

```
x-extend:
  schema: settings.yaml
type: object
properties:
  orphanResources:
    type: array
    default:
      - kind: Job
        version: batch/v1
      - kind: ConfigMap
        version: v1
    items:
      type: object
      required: [kind, version]
      properties:
        kind:
          type: string
        version:
          type: string
```

Each item of the list contains the following fields:

- `kind`: Kind of the objects, for example, `Job`.
- `version`: API version of the kind: `<GROUP>/<VERSION>`, for example, `batch/v1`, or only `<VERSION>`, for example, `v1`. A version without a group matches the kind in every API group that serves it at this version, so `kind: Job` with `version: v1` matches the `batch/v1` Jobs too. To match the kind in one group only, specify the group.

Declare the field in `openapi/values.yaml` rather than in `openapi/settings.yaml`, so that users can't change the list in `spec.settings`. DP reads the list from the values of the current package version when the Application is deleted. If earlier versions of the package created objects of other kinds, keep these kinds in the list.

DP deletes the objects of the listed kinds that are located in the namespace of the Application and carry the `packages.deckhouse.io/instance: <INSTANCE_NAME>` label. The object names don't matter. DP adds the label to every object rendered from the templates, including Helm hooks (for details, refer to "Templates"). Objects created by workloads and hooks don't get the label automatically, so set it yourself. For example, a hook sets the label to the instance name:

```
input.PatchCollector.CreateOrUpdate(&corev1.ConfigMap{
   TypeMeta: metav1.TypeMeta{APIVersion: "v1", Kind: "ConfigMap"},
   ObjectMeta: metav1.ObjectMeta{
      Name:  "state",
      Labels: map[string]string{"packages.deckhouse.io/instance":
input.Instance.Name()},
   },
   Data: map[string]string{"instance": input.Instance.Name()},
})
```

A workload can get the instance name from the templates (`.Application.Instance.Name`), for example, in an environment variable.

DP deletes the objects after the Helm release is uninstalled and the hooks are stopped, so that the application can't create them again. Dependent objects are deleted in the background: for example, the Kubernetes garbage collector deletes the Pods of a deleted Job, so their kind doesn't need to be listed. DP doesn't wait until the objects are gone: an object with finalizers stays in the cluster until its finalizers are removed, even after the Application is deleted.

DP doesn't delete:

- Objects with the `helm.sh/resource-policy: keep` annotation, even if the `werf.io/resource-policy` annotation is set. Other annotations don't protect an object: DP deletes objects whose `werf.io/resource-policy` annotation contains `keep` or `skip-delete` and objects with the `werf.io/ownership: anyone` annotation, for example, the Job from "Resource rendered only on first install".
- Objects that are already being deleted.
- Objects of cluster-wide kinds and of kinds that the cluster doesn't serve, for example, custom resources whose CustomResourceDefinition is not installed. DP skips such kinds and writes a warning to its log.

DP deletes orphan resources only when the Application is deleted. When the application is suspended or its package version changes, the objects stay, because the application can use them again. If the package of the application isn't loaded when the Application is deleted, for example, because DP failed to download the package bundle, DP uninstalls the Helm release but doesn't delete orphan resources.

If DP fails to list or delete the objects, for example, when the API server is unavailable, DP deletes the rest of the objects and retries the step. If the `orphanResources` value is not a list of kinds, DP writes an error to its log and removes the Application without deleting orphan resources.

DP writes every deleted object, every object with the `helm.sh/resource-policy: keep` annotation and every skipped kind to its log: search the log for `orphan`. While DP retries the step, the queue of the application contains the `Purge` task with the error (for details, refer to "State in DP").

Resource restoration

DP remembers the resources rendered from the templates and checks every 4–5 minutes that all of them exist in the cluster. If a resource is missing, DP starts a reconciliation run, and the resource is created again.

DP checks only whether the resources exist, not their contents. The check is paused while hooks run and is disabled in maintenance mode.

Deployment tracking

When DP applies the templates, Nelm waits until the resources become ready. The waiting is controlled by the tracking annotations. Pod logs are not collected, so the log annotations have no effect.

The progress of the current apply is published in the `status.tracking` field of the Application: a list of Nelm operations with their states and dependencies.

One apply is limited to 30 minutes. When the limit is exceeded, DP cancels the apply, and the condition message names up to five resources DP was waiting for, for example:

```
install nelm release 'my-namespace.myapp': apply timed out after 30m0s, waiting for Job/d8a-myapp-migrate
```

Whether the application is running after the apply is determined from its Deployments and StatefulSets (see Workload health).

Maintenance mode

To debug or tweak an installed application manually, switch it to maintenance mode:

```
spec:
  maintenance: NoResourceReconciliation
```

In maintenance mode:

- DP stops applying the templates: changes of settings and values are not reflected in the resources. Hooks keep running.
- Deleted resources are not restored.
- DP labels the resources rendered from the templates with `maintenance.deckhouse.io/no-resource-reconciliation`, and the admission policies stop protecting them, so the resources can be changed manually. To add the label, DP applies the templates once when the mode is enabled.
- The `ApplicationIsInMaintenanceMode` alert fires.

To return the application to normal operation, remove the `spec.maintenance` field. DP applies the templates, which restores the managed state of the resources and removes the label.

Debugging

Application status

The following fields of the Application status are useful when you develop a package:

Field	Description
`status.summary`	High-level state of the application, the reason, and a hint on what to do
`status.conditions`	Detailed conditions (for details, refer to “Installing and managing applications”)
`status.currentVersion.version`	Installed package version
`status.lastAppliedConfiguration`	Effective settings of the last successful apply
`status.urls`	Application endpoints
`status.tracking`	Nelm progress report of the current or the last apply

Example:

```
d8 k -n <NAMESPACE> get applications <INSTANCE_NAME> -o jsonpath='{.status.summary}'
```

State in DP

The `deckhouse-controller packages` command shows the state of the application in DP. The package name in this command is formatted as `<NAMESPACE>.<INSTANCE_NAME>`. Run the command in the `deckhouse` container of the DP leader pod.

> ⚠ The output contains values and container registry credentials. Only cluster administrators should have access to it.

- To render the templates with the current values, run the following command:

  ```
  d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller packages render <NAMESPACE>.<INSTANCE_NAME>
  ```

- To show the loaded package (path, definition, image digests, repository, values, hooks, and internal conditions), run the following command:

  ```
  d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller packages dump --name <NAMESPACE>.<INSTANCE_NAME>
  ```

- To show the snapshots of the hook Kubernetes bindings, run the following command:

  ```
  d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller packages snapshots <NAMESPACE>.<INSTANCE_NAME>
  ```

- To show the task queues of the application, run the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller packages queue dump --name <NAMESPACE>.<INSTANCE_NAME>
```

- To show the scheduler state of the application (for example, which requirement is not met), run the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller packages scheduler dump --name <NAMESPACE>.<INSTANCE_NAME>
```

Logs

DP writes the application processing events, including the output of hooks, to its log:

```
d8 k -n d8-system logs svc/deckhouse-leader -c deckhouse | grep '<NAMESPACE>.<INSTANCE_NAME>'
```

Additional resources

- Global custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Disaster resilience

Disaster resilience is the ability of infrastructure based on Deckhouse Platform (DP) to remain operational in the event of large-scale failures. It is achieved through distributed deployment, automatic traffic switching, and replication of critical components.

DP implements two approaches to disaster resilience:

- Geo-distribution, which is a distribution of infrastructure components across different availability zones (Multi-AZ) or regions (Multi-Region). This approach helps reduce the impact of infrastructure failures on application availability. For details, refer to Geo-distribution.
- Geo-reserving, which is based on using multiple independent clusters combined into a multi-cluster system. If one of the clusters becomes unavailable, traffic can be automatically redirected to another. For details, refer to Geo-reserving.

Both approaches require configuring network connectivity between nodes and regions, traffic balancing, and proper storage organization. The choice of an architecture depends on the specifics of applications and infrastructure constraints.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Geo-distribution

Geo-distribution, in the context of disaster resilience in Deckhouse Platform (DP), is an architectural approach that ensures continuous cluster operation by distributing its components across multiple availability zones (Multi-AZ) or regions (Multi-Region). This approach minimizes the risks of incidents of various scales.

> ⚠ When implementing geo-distribution, it’s important to ensure network connectivity between cluster nodes. This is the responsibility of the platform administrator.

Using multiple availability zones (Multi-AZ)

DP lets you distribute cluster nodes across availability zones. This approach increases application resilience to infrastructure-wide failures and ensures the applications remain functional even if one of the zones in a data center or cloud becomes unavailable.

An **availability zone (AZ)** is the basic unit of a disaster-resilient architecture. It represents a logical group of one or several data centers located within a few kilometers of each other. An AZ can be viewed as an isolated unit (a virtual data center), even though it's a physically distributed infrastructure. Availability zones are part of regions.

Availability zone features

Availability zones are defined by the following features:

- Sufficient physical separation between data centers within the same AZ (usually ≤10 km) to protect from localized incidents (such as floods or fires), while maintaining low network latency.
- Independent power supply for components within the zone (different substations, diesel generators, etc.).
- Use of different providers and network routes by components within the zone.
- Relatively low latency within the zone.

Distributing nodes across zones

With most cloud providers, zone distribution happens automatically. In non-cloud environments, administrators must manually label nodes to indicate the target zone.

For the node labeling process and how to add nodes to groups, refer to "Node management".

Using multiple regions (Multi-Region)

DP supports distributing cluster nodes across different regions. This approach allows applications to continue running even in the event of a complete outage of a data center or cloud provider.

A **region** is an independent geographic area that includes several (typically at least three) isolated availability zones (AZs). All zones within a region are united into a single logical infrastructure unit with shared management services. A region is a key element in the global distributed architecture used by cloud providers.

> **i** Using multiple regions (Multi-Region) is usually appropriate only when applications can work independently in each region (without reaching out to others). This is due to the relatively high latency between regions. Typical use cases include hosting CDN cache servers or distributed monitoring nodes.

Region features

Regions are defined by the following features:

- Latency differences within a region and between regions. Within a region, latency between AZs can be around 5 ms. Between regions, it can be between 50 and 200 ms.
- Geographic distance. Regions may be widely dispersed. For example, the distance between `europe-west3` (Frankfurt) and `us-east-1` (Virginia) is around 6500 km.
- Physical safety. Providers try to place regions outside of seismically or politically unstable areas.
- Energy redundancy. Regions are usually connected to multiple national power grids (for example, `me-central-1` (UAE) uses 4 independent power sources).

Distributing nodes across regions

Distributing nodes across regions requires the following:

- Labeling each node to indicate the region it belongs to.
- Providing stable communication channels between regions.

For the node labeling process and how to add nodes to groups, refer to "Node management".

Balancing incoming traffic

In a geo-distributed cluster, traffic balancing can be handled either by the cloud provider or by internal load balancers (such as MetalLB). The specific implementation depends on the infrastructure and must be configured by the administrator. For details on traffic balancing options and configuration, refer to Network.

Storage organization

Storage system organization and configuration in a geo-distributed cluster are handled by the administrator. DP supports various storage options and the choice depends on the specific project needs. For details on

supported storage systems, their features and configuration, refer to Storage.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Geo-reserving

Geo-reserving, in the context of disaster resilience in Deckhouse Platform (DP), is a method of increasing fault tolerance by combining multiple independent Kubernetes clusters into a single multi-cluster system (multi-cluster) distributed across different geographic locations.

DP offers several tools to unite multiple clusters into a multi-cluster. This approach ensures automatic routing of both external and internal traffic to another cluster with an available copy of the application, in case it becomes unavailable in the original cluster.

Clusters can be combined using:

- The built-in Istio-based Service Mesh.
- DP's networking features powered by Cilium.

Enabling multi-cluster functionality requires deploying one DP cluster per region and then connecting them using a declarative API. It is important that the application can run in parallel across different regions, and that there is a stable network connection between the regions. A properly configured system ensures global disaster resilience for the application, guaranteeing uninterrupted operation even if an entire region or cloud becomes unavailable.

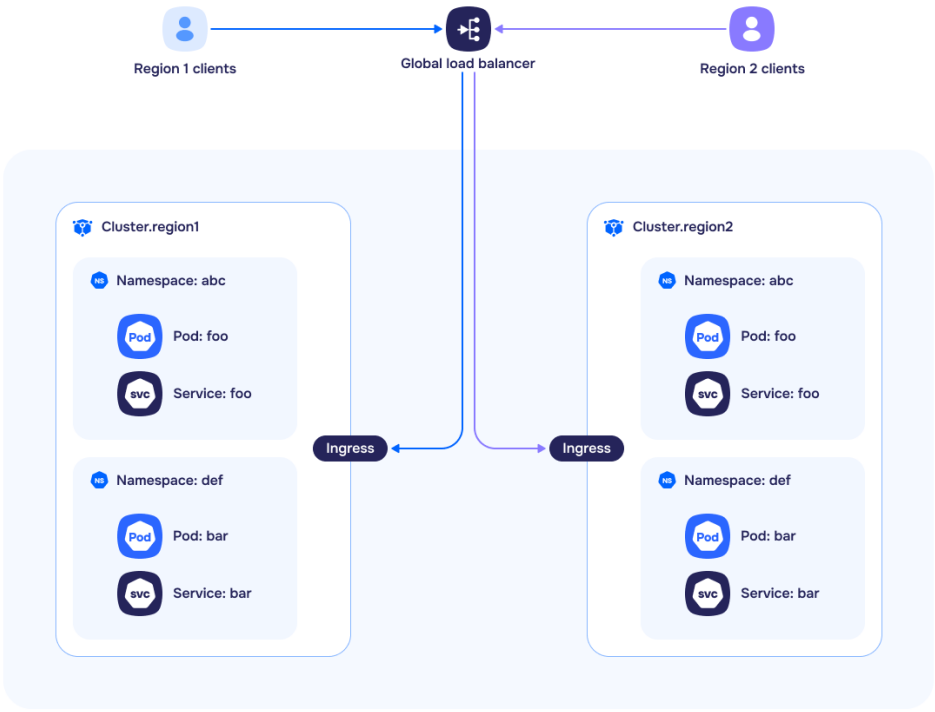
For the process of joining multiple clusters into a multi-cluster, refer to Inter-cluster cooperation.

Balancing incoming traffic in a multi-cluster

To ensure disaster resilience in a multi-cluster, incoming traffic can be balanced by an external load balancer using one of the following methods: `active-active` or `active-standby`. The balancing configuration is handled by the cluster administrator.

Balancing traffic via active-active method

Following the `active-active` method, the external load balancer distributes traffic across multiple active clusters, for example, based on geographic proximity or current workload.

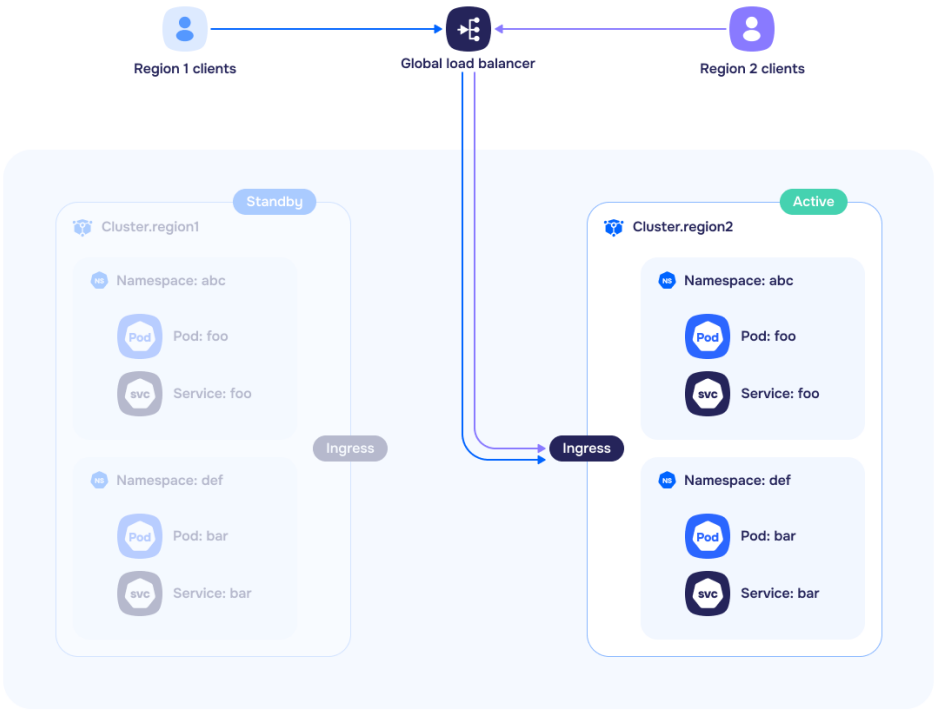


Key aspects of `active-active` balancing:

- All clusters in the multi-cluster process traffic simultaneously.
- Traffic is routed to the nearest and least loaded cluster, or according to other defined rules.
- Shared data is replicated across clusters.
- In the event of a failure in one cluster, traffic is automatically rerouted to the others (automatic failover).

Balancing traffic via active-standby method

Following the `active-standby` method, the external load balancer routes traffic only to the currently active cluster. The standby cluster remains idle and receives traffic only if the primary cluster becomes unavailable.



Key aspects of `active-standby` balancing:

- The active cluster receives all traffic.
- The standby cluster remains idle until the primary cluster fails.
- Traffic failover to the standby cluster is performed automatically upon detection of issues.

Storage organization in a multi-cluster

Storage system organization and configuration in a multi-cluster are handled by the administrator. DP supports various storage systems. The choice of a suitable solution depends on requirements of performance, fault tolerance, and data synchronization methods. For details on supported storage systems, their features and configuration, refer to Storage.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Updating

Releases

A release is any published version of Deckhouse Platform (DP). Each release is distributed through release channels with defined delays. DP publishes two types of releases:

- **Patch version** (for example, from `0.0.1` to `0.0.2`): Includes bug fixes and is published when needed.
- **Minor version** (for example, from `0.0.1` to `0.1.0`): Includes new features and is published every 3–4 weeks.

Release channels

> **i** Up-to-date information about DP versions available on different release channels is available at releases.deckhouse.io.

DP uses **five release channels** to gradually roll out new versions. Each new DP version is first published to the **Alpha** channel and then gradually moves to **Rock Solid**. Updates in less stable channels are made available to a limited number of users, which allows to detect and resolve potential issues before they affect production environments.

- **Rock Solid**: The most stable release channel. Ideal for clusters requiring maximum reliability. Updates are published at least 30 days after the release.
- **Stable**: A stable channel suitable for production clusters. Updates are published at least 14 days after the release.
- **Early Access**: The recommended release channel if you're unsure which one to choose. Suitable for actively evolving clusters (for example, those launching or refining new applications). Updates are published at least 7 days after the release.
- **Beta**: Intended for development clusters, similar to the Alpha channel. Receives versions that have already been tested in Alpha.
- **Alpha**: The least stable channel with the most frequent updates. Targeted at development clusters with a small number of developers. Versions appear immediately after release.

Switching to a more stable channel

When switching to a more stable channel (for example, from `Alpha` to `EarlyAccess`):

1. DP fetches release data from the `EarlyAccess` channel.
2. It compares this data with existing DeckhouseRelease custom resources in the cluster.
   - If the cluster contains newer releases with `Pending` status (not yet applied), they will be **removed**, since they haven't been published to the new channel.
   - If newer releases have already been marked as `Deployed` (installed successfully), the switch won't take effect immediately. DP will remain on the current release until a newer version becomes available in the `EarlyAccess` channel.

Switching to a less stable channel

When switching to a less stable channel (for example, from `EarlyAccess` to `Alpha`):

1. DP fetches release data from the `Alpha` channel.
2. It compares this data with existing DeckhouseRelease custom resources.
3. It applies the update according to the configured update parameters.

Control plane updates

In DP, the control plane update process is highly automated and safe for both single-master and multi-master clusters. While brief interruptions in API server availability may occur, they do not affect the operation of applications running in the cluster. In most cases, no additional maintenance window is required.

DP supports the latest five minor Kubernetes versions. A full list of supported versions is available in the corresponding table.

Patch version updates

Patch updates to control plane components (within the same minor version, for example, from `1.35.3` to `1.35.5`) are applied automatically together with DP updates. Users cannot manage patch updates manually. The process is fully automated by the platform.

Automatic minor version updates

To automatically update the control plane to a new minor version (for example, from `1.34.*` to `1.35.*`), specify `kubernetesVersion: Default` in the `control-plane-manager` ModuleConfig. DP will select the default Kubernetes version at the time of the update.

Manual minor version updates

To manually update the control plane to a new minor version (for example, from `1.34.*` to `1.35.*`), specify the target version in the `kubernetesVersion` parameter of the `control-plane-manager` module. For example, `kubernetesVersion: 1.35`.

```
d8 k edit mc control-plane-manager
```

The command opens the `control-plane-manager` ModuleConfig for editing. After the change is saved, DP starts the upgrade to the specified version. To track the upgrade progress, check the Kubernetes version in

the output of the node description command:

```
d8 k get nodes
```

Process of control plane component updates

1. The control-plane-manager module modifies the manifests of core components (`apiserver`, `controller-manager`, `scheduler`, etc.).
2. The updated components are deployed on all master nodes.

When performing a minor version upgrade:

- If bumping for more than one minor version (for example, from `1.33` to `1.35`), the update happens in stages: `1.33` → `1.34` → `1.35`.
- At each step, the control plane is updated first, followed by the kubelet upgrade on the nodes.

When performing a downgrade:

- Only a single minor version downgrade is supported from the highest version ever used in the cluster.
- The downgrade occurs in reverse order. Kubelet components on the nodes are downgraded first, followed by the control plane components.

After the control plane has been updated, node updates begin:

1. The Bashible API Server updates the kubelet version in the scripts for all NodeGroups.
2. Nodes in different NodeGroups are updated in parallel, but within each NodeGroup, they are updated sequentially (one by one).
3. Each NodeGroup's update settings are taken into account:
   - Manual or automatic update mode.
   - Update windows.
   - Whether Pods should be evicted before updating, etc.
4. One or more candidate nodes that meet all update conditions are selected and updated.
5. The process repeats until all nodes in all groups are updated.
6. When all nodes are updated, the update process is complete.

Retrieving the changelog

Each new version of DP includes a *changelog*, which is a detailed list of changes, including new features, bug fixes, component updates, and important compatibility notes.

You can find the changelog for a specific DP version in the Deckhouse release list on GitHub.

A summary of key changes, component version updates, and which cluster components will be restarted is included in the description of the zero patch release: example for DP v1.68.

Changelog contents

The changelog includes four sections:

- **Know before update**: Critical information to consider before updating. Includes compatibility requirements, pre-update actions, and possible cluster impacts.
- **Features**: Highlights of key improvements and new features introduced in the release.
- **Fixes**: Minor changes, security updates, and performance improvements.
- **Chore**: Technical updates, such as dependency updates, refactoring, build and test pipeline changes, and vulnerability fixes.

Minor versions and zero patch releases

All major changes are listed in the **zero patch release** (for example, `v1.68.0` for the DP `v1.68`). Before updating to a new minor version (for example, `v1.68`):

1. Review the changelog for the corresponding version.
2. Check if the changes affect your infrastructure.
3. Adjust the cluster configuration if needed.

Checking dependencies before update

Before applying a new release, DP checks the cluster for potential issues. If any of the following incompatibilities are detected, the update is aborted:

- Unsupported Kubernetes version.
- When `kubernetesVersion: Automatic` is enabled, the update is aborted if:
  - The release introduces a new default Kubernetes version;
  - Monitoring is enabled;
  - The cluster contains resources that are deprecated in the new version.
- The installed Ingress controller version is incompatible with the new release.

- Nodes are running outdated or unsupported operating systems.
- The cluster has an enabled module that is `deprecated` or has been removed in the new release.

Additional resources

- Module `control-plane-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse subsystem

This subsection describes the architecture of the modules that are part of the Deckhouse subsystem of Deckhouse Platform (DP).

The Deckhouse subsystem includes the following modules:

- `deckhouse`: Deckhouse controller.
- `console`: Deckhouse web UI.
- `deckhouse-tools`: Provides a web interface for downloading the `d8` CLI utility.
- `documentation`: Provides a web interface with documentation corresponding to the running DP version.
- `registry`: Manages the configuration of DP components responsible for working with the container registry and provides an internal image storage.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Web interface modules

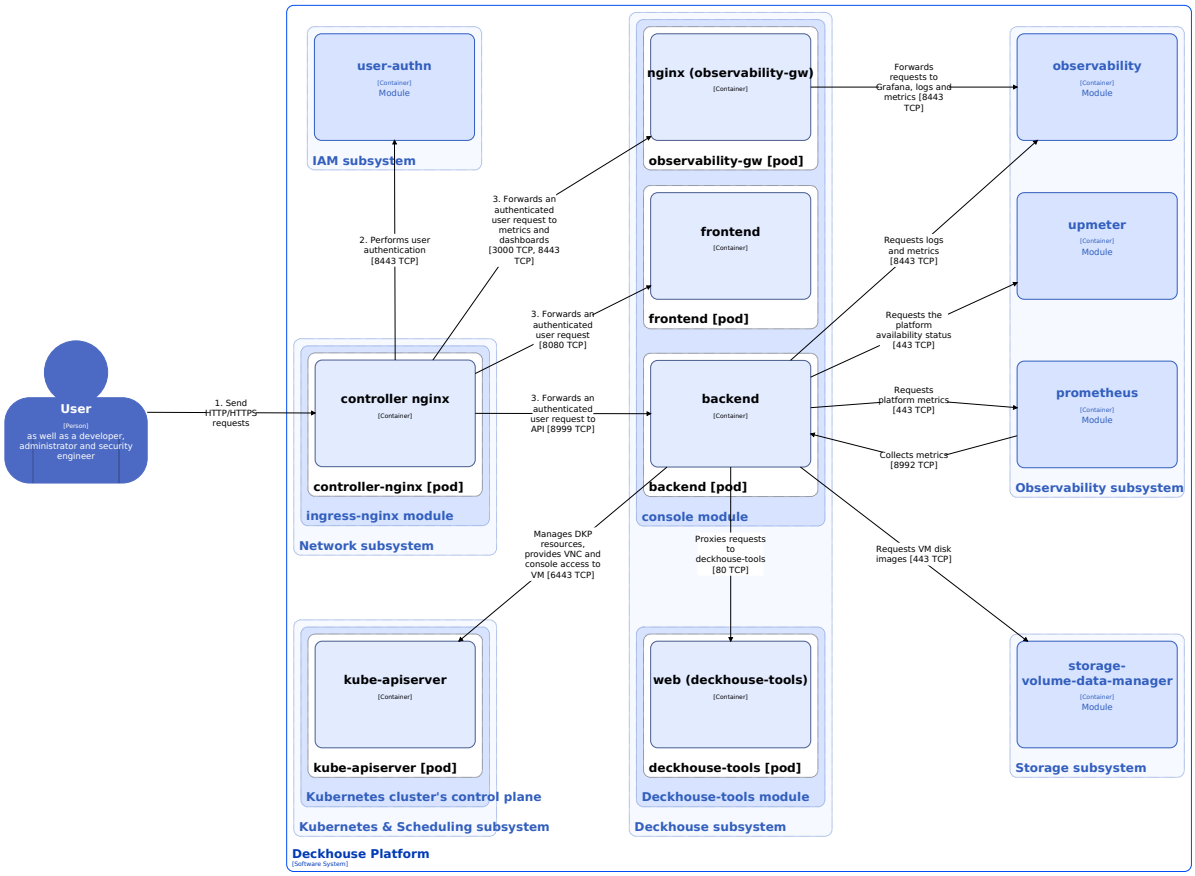
Console module

The `console` module implements the web interface of Deckhouse Platform (DP), simplifying platform management and enabling the system state monitoring.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `console` module and its interaction with other DP components are shown in the following diagram:



> **i** The numbers in the diagram indicate the order in which the user's request passes through the `frontend`, `backend`, and `nginx` components.
>
> At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module. For more information about the architecture of the `user-authn` module, refer to the corresponding documentation section.

Module components

The module consists of the following components:

1. **Frontend**: Consists of a single **frontend** container and provides the web interface for DP users and administrators.
2. **Backend**: Consists of a single **backend** container and implements an API interface that provides the following capabilities:
   - Retrieving, creating, deleting, and modifying DP resources according to the user's permissions.
   - Generating a kubeconfig with the user's profile.
   - Determining the environment in which the DP cluster is deployed.
   - Detecting DP and Kubernetes versions.
   - Loading metrics and logs.
   - Loading platform availability information.
   - Downloading VM disks (when the `virtualization` and `storage-volume-data-manager` modules are enabled).
3. **Observability-gw**: Consists of a single **nginx** container and proxies requests to Grafana to embed dashboards into the platform's main web UI, as well as to work with metrics and logs for the selected project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Establishing VM connections via console and VNC.
   - Creating, deleting, modifying, and tracking DP resources.
2. **Upmeter**: Retrieves DP availability information.

3. **Deckhouse-tools**: Forwards requests to download the Deckhouse CLI utility.
4. **Prometheus**: Retrieves system metrics for the platform, such as CPU and RAM usage.
5. **Observability**: Retrieves metrics and logs for the selected project.
6. **Storage-volume-data-manager**: Exports VM disk images.

The following external component interacts with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

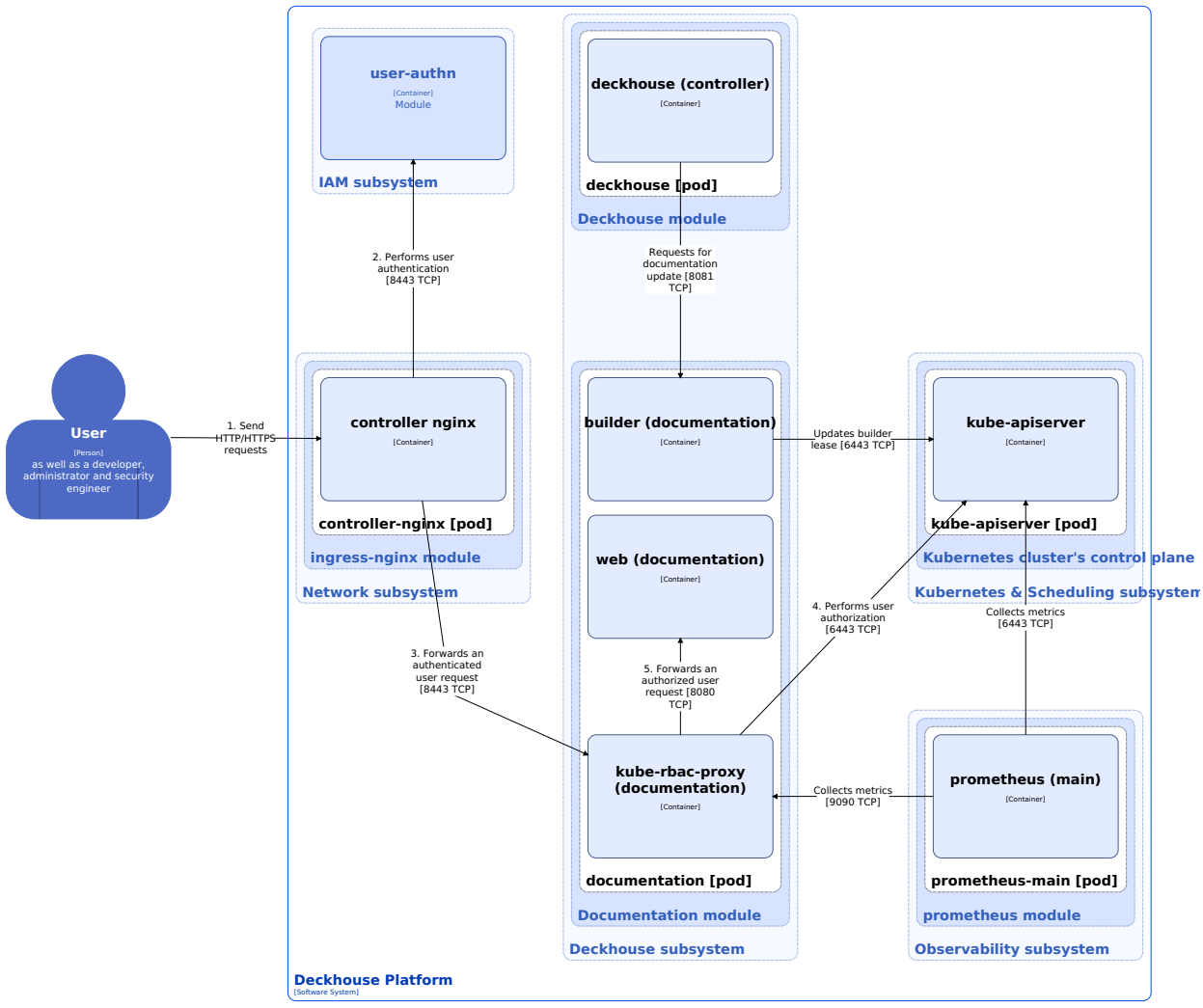
Documentation module

The `documentation` module provides a web interface for documentation corresponding to the running version of DP.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `documentation` module and its interaction with other DP components are shown in the following diagram:



> **i** The numbers in the diagram indicate the order in which the user's request passes through the `web` component:
>
> - At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module.
> - At steps 4 and 5, user authorization based on the Kubernetes RBAC is performed to provide secure access.

Module components

The module consists of the following component:

- **Documentation**: A component that provides the documentation web interface.

  It consists of the following containers:

  - **web**: Main container.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container. It is an open source project.
  - **builder**: Sidecar container that dynamically extends the documentation when new DP modules are installed. The Hugo static site generator is used to render and generate the up-to-date site content.

    The **builder** container automatically creates and updates a Kubernetes Lease resource, placing an endpoint for documentation updates. This endpoint is used by the `deckhouse` module controller to initiate documentation updates when modules are updated or installed. This ensures that changes are promptly rendered in the web interface.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Creates and updates the Lease resource.
  - Authorizes requests to the documentation web interface.

The following external components interact with the module:

1. **Deckhouse**: Sends requests to update the documentation when the set of modules changes.
2. **Prometheus**: Collects module metrics.
3. **Controller nginx**: Forwards external user requests to the module web interface.

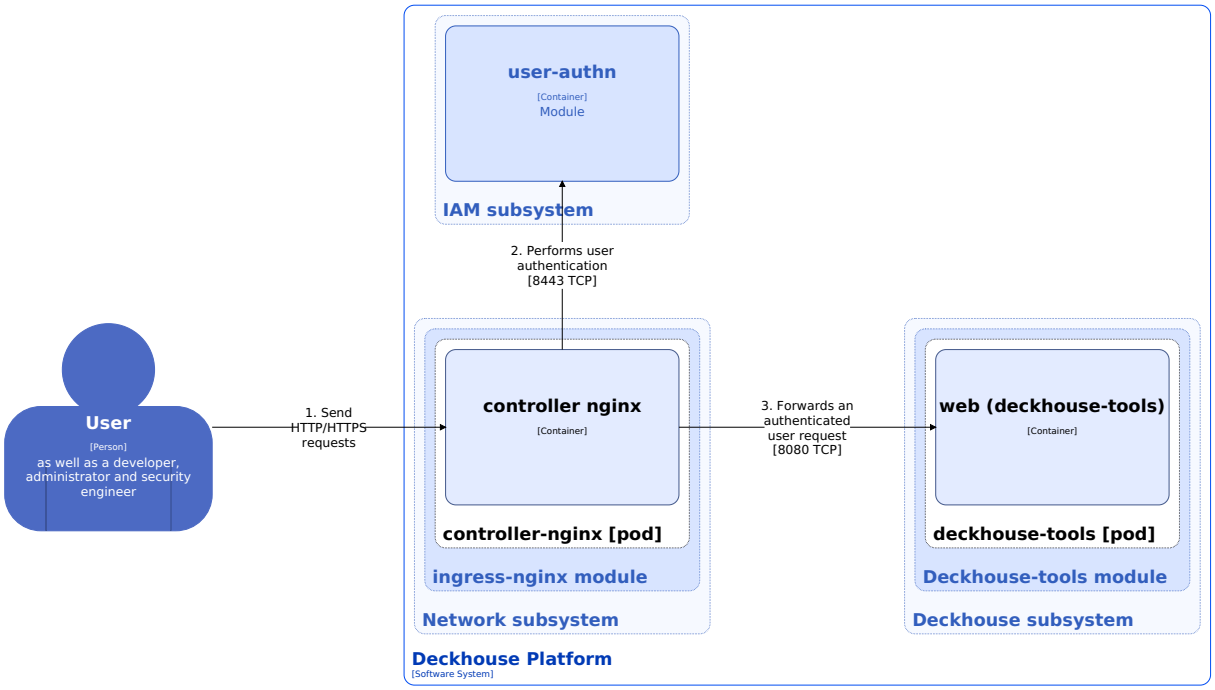
Deckhouse-tools module

The `deckhouse-tools` module provides a web interface with links to download the Deckhouse CLI utility for various operating systems.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `deckhouse-tools` module and its interaction with other DP components are shown in the following diagram:



> The numbers in the diagram indicate the order in which the user's request passes through the `web` component.
>
> At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module.

Module components

The module consists of the following component:

- **Deckhouse-tools**: Consists of a single **web** container and provides a web interface with links to download the Deckhouse CLI utility.

Module interactions

The following external component interacts with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `console` documentation
- Module `deckhouse-tools` documentation
- Module `prometheus` documentation
- Module `storage-volume-data-manager` documentation
- Module `upmeter` documentation
- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse module

The `deckhouse` module implements the core of Deckhouse Platform (DP), performing the following operations:

- Platform updates
- Module configuration management
- Module installation and updates

- Module documentation build triggering
- Validation of custom resources managed by DP modules

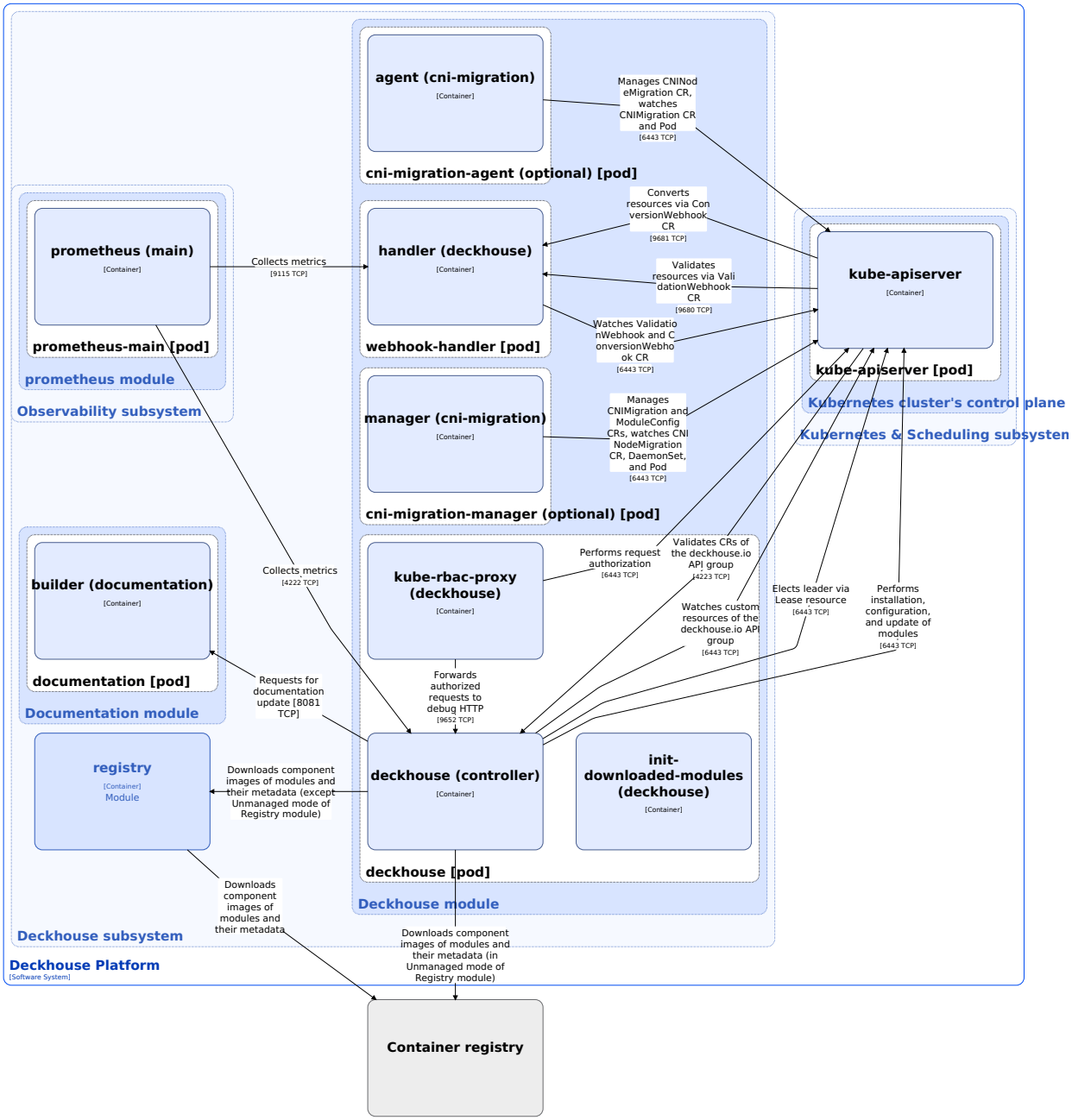
The module manages the following custom resources in the `deckhouse.io` API group:

- Module management:
  - Module: Description, status, and publication of module information.
  - ModuleConfig: Description of user-defined module settings.
  - ModulePullOverride: Description of module version selection overrides.
  - ModuleRelease: Description, publication, and tracking of module releases.
  - ModuleSettingsDefinition: Schema, versions, and transformation rules for module settings.
  - ModuleSource: Description of a module source, repository, or storage.
  - ModuleUpdatePolicy: Rules for module updates and version transition automation.
- Platform management:
  - DeckhouseRelease: An object that defines the DP release (version) and platform update policy.
- Package management (Marketplace):
  - Application: An installed instance of an application package and its settings.
  - ApplicationPackage: Aggregated information about a package: the repositories that provide it, the versions the release channels point to, and the applications that use it.
  - ApplicationPackageVersion: A discovered package version: its metadata, OpenAPI schemas, and requirements.
  - PackageRepository: Connection to a container registry with packages and the scan schedule.
  - PackageRepositoryOperation: A scan operation that discovers package versions in a repository.
- Utility management:
  - CNIMigration: Container Network Interface (CNI) migration process, including migration parameters and status.
  - CNINodeMigration: Status and management of CNI migration at the individual node level.
  - ObjectKeeper: A resource that links Kubernetes resources using `ownerReference`.
- Management of custom resources controlled by DP modules:
  - ConversionWebhook: Settings and handlers for resource conversion webhooks.
  - ValidationWebhook: Settings and handlers for resource validation webhooks.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `deckhouse` module and its interaction with other DP components are shown in the following diagram:



Module components

The module consists of the following components:

1. **Deckhouse** (Deployment): A controller that implements platform management operations.

   The controller coordinates platform management tasks using the queueing mechanism.

   The Deckhouse controller can run in standard mode or in hook isolation mode. To enable it, create the `chroot-mode` ConfigMap in the `d8-system` namespace. In isolation mode, shell hooks and module enable scripts run in a chroot environment with a limited set of mounted directories, isolating them from the controller container file system.

   If High Availability (HA) mode is enabled, multiple Deckhouse controller instances are started. To ensure correct behavior, Deckhouse controllers perform leader election using the `deckhouse-leader-election` Lease resource. The controller elected as leader performs all platform management operations.

   In addition, you can set the following parameters using the Deckhouse controller:

Module configuration parameter	Description
`logLevel`	Logging level
`bundle`	Set of modules enabled by default
`releaseChannel`	Release channel

Module configuration parameter	Description
`update.mode`	Update mode
`update.windows.days`	Update windows

For details about module settings, refer to the module documentation section.

It consists of the following containers:

- **init-downloaded-modules**: Init container that prepares the directory structure required for module operations.
- **deckhouse**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container component debug interface.

2. **Webhook-handler** (Deployment): Consists of a single **handler** container and implements a generic webhook for conversion and validation of custom resources managed by DP.

   The component watches ConversionWebhook and ValidationWebhook custom resources and, based on them, generates hook Python files for shell-operator from templates. When `kube-apiserver` sends resource validation or conversion requests, shell-operator runs the required hook and returns the processing result.

3. **Cni-migration-manager** (Deployment): An optional component running on control plane nodes, consisting of a single **manager** container. The component manages the network plugin (CNI) switching process in the DP cluster and records the current state in the CNIMigration custom resource. Migration to Flannel, Simple bridge, and Cilium is supported. For details on switching CNI in the cluster, refer to the corresponding guide.

   > **i** The component is created by the `detect-cni-migration` global hook when the CNIMigration custom resource exists. The CNIMigration resource is created manually by an administrator or by running the `d8 network cni-migration switch --to-cni <target cni>` command.

4. **Cni-migration-agent** (DaemonSet): An optional component running on all cluster nodes, consisting of a single **agent** container. The component watches the CNIMigration custom resource and manages the CNINodeMigration custom resource that reflects migration state for a specific node.

   > **i** The component is created by the `detect-cni-migration` global hook when the CNIMigration custom resource exists. The CNIMigration resource is created manually by an administrator or by running the `d8 network cni-migration switch --to-cni <target cni>` command.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Working with custom resources in the `deckhouse.io` API group.
   - Watching Pod and DaemonSet resources, as well as restarting Pods when the network plugin is changed.
   - Watching resources described in the ObjectKeeper custom resource.
   - Creating and updating Lease resources.
   - Creating, deleting, modifying, and watching resources described in DP modules.
   - Authorizing requests.
2. **Documentation**: Updating documentation when a DP module is added or updated.
3. **Container registry**: Retrieving module component images along with metadata when the `registry` module is installed in `Unmanaged` mode.
4. **`registry` module**: Retrieving module component images along with metadata when the `registry` module is installed in one of the following modes: `Direct`, `Proxy`, or `Local`.

The module is interacted with by the following external components:

- **Kube-apiserver**: Validation and conversion of DP custom resources.
- **Prometheus-main**: Collecting metrics from the `deckhouse` and `webhook-handler` containers.

Additional resources

- Global custom resources
- Module `deckhouse` documentation
- Module `documentation` documentation

- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Queuing mechanism

The `deckhouse` module implements the core of Deckhouse Platform (DP), performing various platform management operations using a queueing mechanism. For more information about the module architecture, refer to the corresponding documentation section.

The Deckhouse controller implements addon-operator and marketplace queues.

Addon-operator queues

The **addon-operator queues** are the primary processing mechanism for built-in and external DP modules. The queue is implemented in shell-operator and extended with addon-operator task types. The Deckhouse controller synchronizes ModuleConfig custom resources and updates global or module values for addon-operator.

Task types:

Task	Function
GlobalHookRun	Runs global hooks (`onStartup`, `beforeAll`, `afterAll`, `kubernetes`, `schedule`)
GlobalHookEnableKubernetesBindings	Enables Kubernetes monitors from global hooks
GlobalHookWaitKubernetesSynchronization	Blocks the queue until global hooks with `executeHookOnSynchronization: true` complete
GlobalHookEnableScheduleBindings	Registers cron tasks in the addon-operator scheduler
DiscoverHelmReleases	Finds “extra” Helm releases after the first `converge`
ApplyKubeConfigValues	Applies changes from ModuleConfig
ConvergeModules	Full `converge` cycle for all modules
ModuleRun	Configures or updates a module, including subtasks: `onStartup` -> `sync` -> `beforeHelm` -> `helm` -> `afterHelm`
ParallelModuleRun	Launches modules in batch and in parallel
ModuleDelete	Removes a module (`helm delete`, `afterDeleteHelm`)
ModuleHookRun	Runs a module hook on event
ModuleEnsureCRDs	Installs module CRDs
ModulePurge	Removes unknown Helm release

Addon-operator queue types:

Queue	Name	Purpose
Main	`main`	Runs global hooks at startup, installs critical modules, configures and deletes modules
Parallel	`parallel_queue_0` ... `parallel_queue_19`	Parallel ModuleRun with module dependencies taken into account (20 queues)
Hook queues	From task config (hook)	Tasks for a specific module and global hooks

Each queue is a separate pipeline with one worker. The queue has the following properties:

- Tasks can be inserted both at the tail (`AddLast`) and at the head (`AddFirst`) of the queue.
- Tasks are executed from the head of the queue.
- Multiphase operations are supported with the following operations:
  - `AddHeadTasks`: Insert subtasks before the current task.
  - `AddTailTasks`: Insert a subtask at the end of the queue after the current task succeeds.
  - `AddAfterTasks`: Insert a subtask right after the current task.
- A task is executed until success unless `allowFailure: true` is set in task parameters.
- On error, exponential restart (backoff) is applied, starting with a 5-second delay between retries.

> ⚠ If a task cannot complete successfully and `allowFailure: true` is not set in its parameters, that task blocks the queue it runs in. Tasks in different processing queues do not block each other.

At startup, the Deckhouse controller creates `main` and `parallel_queue_0..19` queues and adds the following tasks to `main` in this order:

- GlobalHookRun (onStartup): For each global hook.
- GlobalHookEnableScheduleBindings: To enable cron scheduling.
- GlobalHookEnableKubernetesBindings: To enable global tasks that watch Kubernetes resources.
- GlobalHookWaitKubernetesSynchronization.
- ConvergeModules (OperatorStartup): First converge of all modules.

After ConvergeModules, the controller adds the DiscoverHelmReleases task to clean up unknown Helm releases.

The module processing order in the ConvergeModules task is determined by several attributes:

- Module criticality: The `critical` parameter in the module `module.yaml` configuration.
- Module *weight*: Numeric module processing order. The higher the number, the later the module is processed. The weight is taken from the `weight` parameter in module `module.yaml`; if missing or set to 0, the default weight 900 is used. If no module config file exists, the weight is taken from the numeric prefix of the module directory name (for example, `040-node-manager` means weight 40). If the weight cannot be obtained from the directory name, weight 100 is used.
- Module dependencies: A list of modules that must be installed before the current module.

Based on these attributes, the Deckhouse controller scheduler defines processing order according to the following principles:

- For critical modules, module weight is considered in ascending order and tasks are put into the `main` queue.
- For non-critical modules, module weight is not considered and tasks are put into `parallel_queue_0..19`.
- For all modules, dependencies on other modules are considered.

If critical modules can be processed in parallel, the Deckhouse controller scheduler places the ParallelModuleRun task into the `main` queue with the list of modules. The ParallelModuleRun task starts a ModuleRun task for each module in `parallel_queue_0..19` and waits for completion, thereby blocking the `main` queue. If an error occurs while processing a ModuleRun task, the scheduler moves this task to the end of the queue and starts the next queue task.

The process of installing critical modules is shown in the following diagram:

The process of installing non-critical (functional) modules is shown in the following diagram:

If more than one identical task is added to a queue, all duplicates are removed when such a task starts, as part of deduplication.

To view addon-operator queues, use the `d8 system queue list` command.

Marketplace queues

The **Marketplace queues** are a queue implementation used by Marketplace functionality.

Each queue served by the Deckhouse controller for Marketplace has the following properties:

- FIFO (First In First Out): Defines strict task execution order. The first task in queue is executed first.
- Strict sequential execution of tasks, one task at a time.
- Tasks are started only on events (event-driven), without polling any resources.
- Restart on errors with exponential backoff between retries, starting at 15 seconds and capped at 1 minute, with no limit on retry attempts.
- Supports cascading task cancellation on version changes or package deletion.

Queue types:

Name	Purpose
`{packageName}`	Lifecycle: Deploy, Load, Configure, Enable, Run, Disable, Undeploy
`{packageName}/{hookQueue}`	Running hooks triggered by Kubernetes or schedule events (queue from hook binding)
`{packageName}/{hookQueue}/sync`	Hook synchronization at startup (WaitForSynchronization)

Task types:

Task	Function
Deploy	Downloads or mounts the package image
Load	Parses configuration, creates Application or Module, registers in scheduler
Configure	Applies Application or Module settings using the parameter store
Enable	Enables hooks, performs parameter synchronization, runs OnStartup hook
Run	Runs subtasks when installing Application or Module: `BeforeHelm` -> `helm Upgrade` -> `AfterHelm`
HookRun	Runs a hook on event
HookSync	Performs initial synchronization of Kubernetes binding
Disable	Removes Helm, disables hooks, clears hook queues
Undeploy	Removes the package from disk

Task execution in one queue does not block task execution in another queue.

To view Marketplace queues, run the following command:

⚠ When performing module installation or configuration tasks in addon-operator, the Deckhouse controller pauses Marketplace queue processing.

Additional resources

- Global custom resources
- Module `deckhouse` documentation

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Registry module

The `registry` module manages the registry settings for Deckhouse Platform (DP) components.

The module can operate in the following modes:

- `Direct`: Provides direct access to an external registry via the fixed address `registry.d8-system.svc:5001/system/deckhouse`. This fixed address prevents Deckhouse images from being re-downloaded and components from being restarted when registry parameters are changed. The fixed address is replaced by the actual address of the external registry when sending requests by configuring `mirroring/rewrite` section in the configuration of the containerd component and by using the internal registry-incluster-proxy component that proxies requests to the external registry.
- `Proxy`: Using an internal caching proxy registry that accesses an external registry, with the caching proxy registry running on control-plane (master) nodes. This mode reduces the number of requests to the external registry by caching images. Cached data is stored on the control-plane (master) nodes.

Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` mode.

- `Local`: Using a local internal registry, with the registry running on control-plane (master) nodes. This mode allows the cluster to operate in an isolated environment. All data is stored on the control-plane (master) nodes. Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` and `Proxy` modes.
- `Unmanaged`: Operation without using the internal registry. Access within the cluster is performed directly to the external registry.

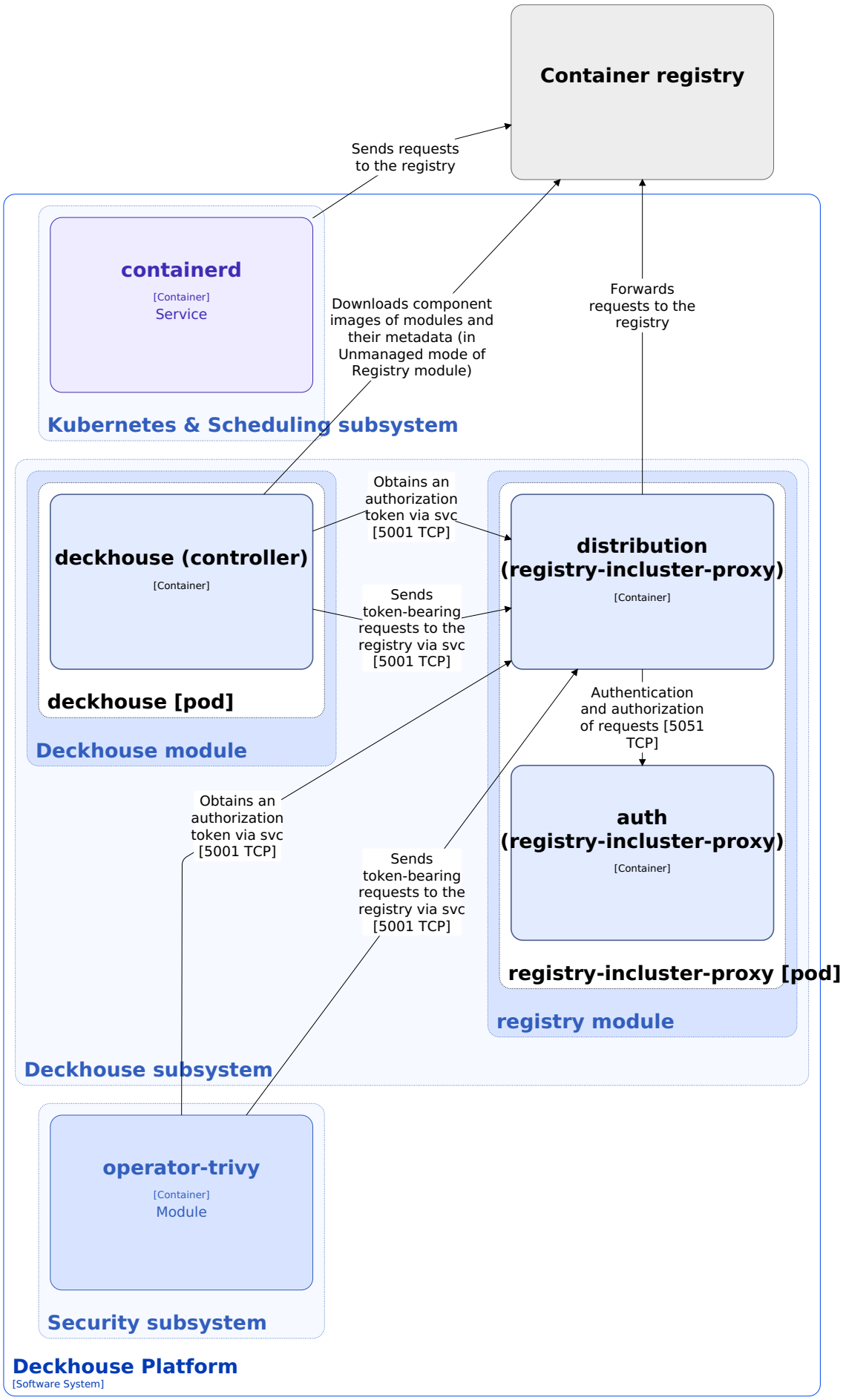
For more details about the configuration and usage examples of the module, refer to the module documentation.

Module architecture

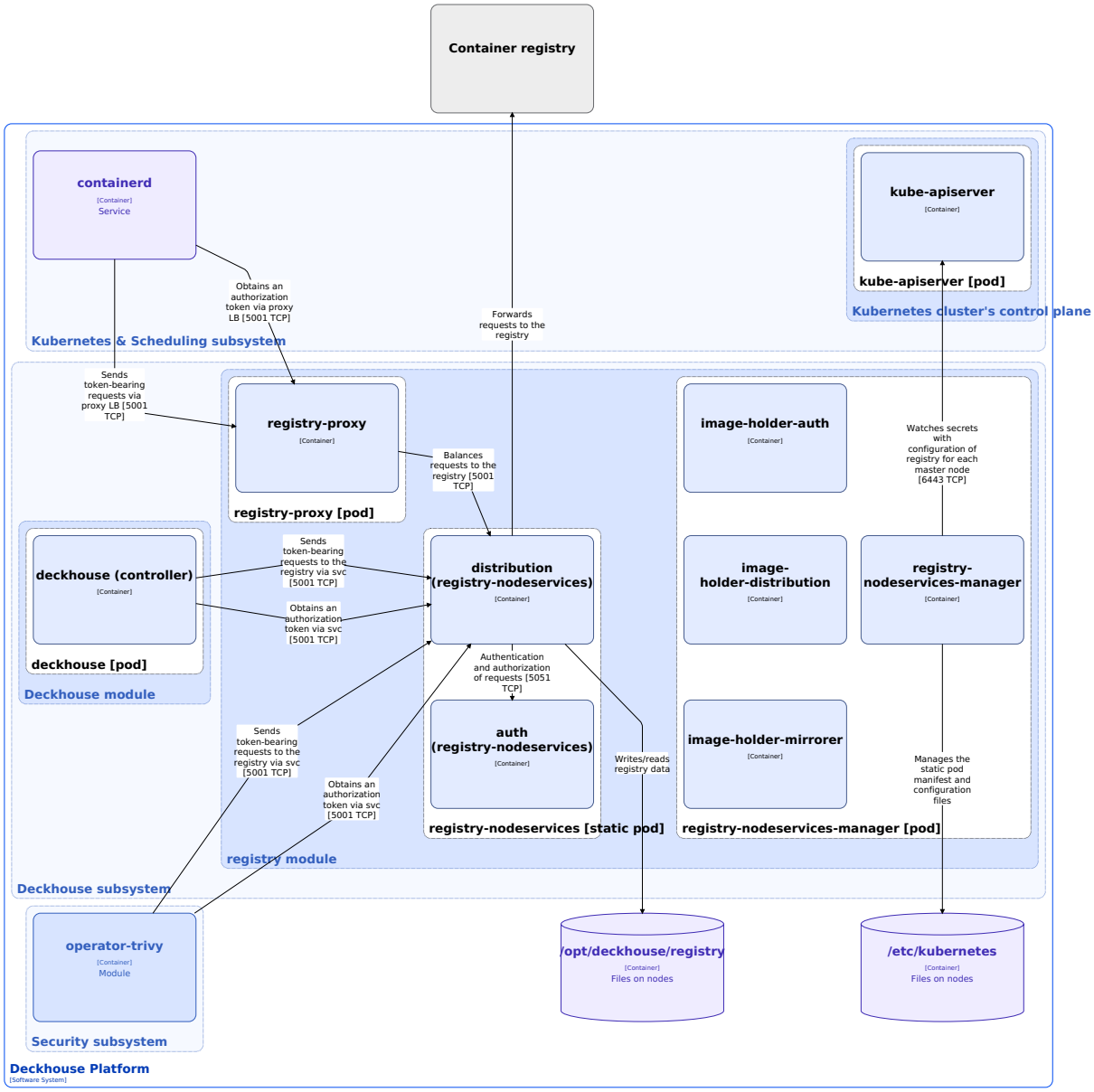
> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `registry` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:

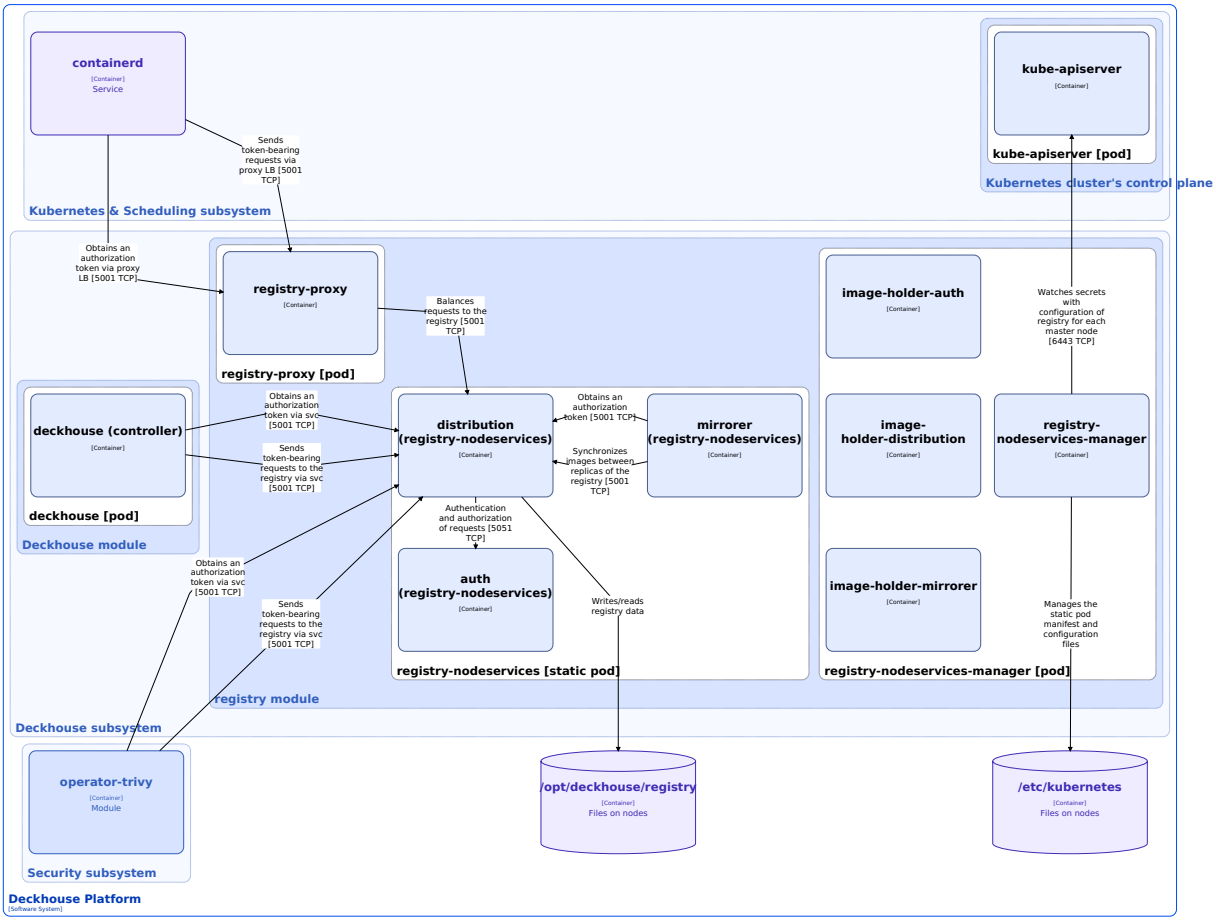
The `registry` module in `Direct` mode:



The `registry` module in `Proxy` mode:



The `registry` module in `Local` mode:



Module components

The module consists of the following components:

1. **Registry-incluster-proxy** (`Direct` mode): A container registry based on Distribution. Distribution is an open-source project that provides a framework for storing and distributing container images and other content using the OCI Distribution Specification. Registry-inclusion-proxy is installed in the cluster as a Deployment, does not store or cache images, but forwards requests to the external registry specified in Direct mode settings in `deckhouse` ModuleConfig.

   Registry-incluster-proxy consists of the following containers:

   - **distribution**: Main container.
   - **auth**: Sidecar container that implements authentication and authorization when accessing the registry. It is an open-source project.

     The principle of operation:

     1. **Request to the authentication service**. When the client tries to access the registry, distribution returns a `401 Unauthorized` HTTP response with the `WWW-Authenticate` header indicating how to authenticate.
     2. **Getting a token**. The client sends a request to the authentication service via the distribution service (for example, `registry.d8-system.svc:5051/auth`) using the values `service` and `scope` from the header `WWW-Authenticate`. The auth service returns an opaque Bearer token that represents authorized client access.
     3. **Token usage**. After receiving the token, the client resends the original request to distribution, including the token in the `Authorization` header.
     4. **Access verification**. Distribution validates the token and the `claims` contained in it, and then begins the image download session.

2. **Registry-nodeservices** (`Proxy` or `Local` modes): A container registry based on Distribution running on control plane (master) nodes. The registry is launched as a static pod, and its lifecycle is managed by the registry-nodeservices-manager component. The data is stored in the directory `/opt/deckhouse/registry` on the control plane (master) nodes.

   Registry-nodeservices consists of the following containers:

- **distribution**: Main container.
- **auth**: Sidecar container for authentication and authorization (described above).
- **mirrorer** (`Local` mode): Sidecar container that synchronizes images between distribution replicas running on control plane (master) nodes.

3. **Registry-proxy** (`Proxy` or `Local` modes): A load balancer installed on each cluster node that provides high availability for registry. The CRI accesses the registry via the load balancer. The settings to access the load balancer are specified in the containerd configuration.

   It consists of a single container:

   - **registry-proxy**: A container built based on a standard NGINX image. The registry-proxy-reloader process is also running in the container and restarts the NGINX load balancer processes when its configuration changes.

4. **Registry-nodeservices-manager** (`Proxy` or `Local` modes): A controller that manages the lifecycle of registry-nodeservices components. Registry-nodeservices-manager performs the following operations:

   - Renders the manifest of the registry-nodeservices static pod from templates and saves it in the `/etc/kubernetes` directory on the control plane (master) nodes. Kubelet detects created manifests and runs registry-nodeservices static pod.
   - Saves configuration files necessary for the operation of registry-nodeservices components in the `/etc/kubernetes/registry` directory on control plane (master) nodes.
   - Deletes the static pod manifest in the `/etc/kubernetes` directory and configuration files in the `/etc/kubernetes/registry` directory, thereby deleting the local container registry, in case the module switches to the `Direct` or `Unmanaged` mode.

   Registry-nodeservices-manager consists of the following containers:

   - **registry-nodeservices-manager**: Main container.
   - A set of sidecar containers used to pre-pull images of registry-nodeservices components. These containers remain paused and serve only as image holders:
     - **image-holder-auth**
     - **image-holder-distribution**
     - **image-holder-mirrorer**

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Checks that the current node is the master.
   - Checks that all containers of the registry-nodeservices static pod are pulled and status of registry-nodeservices pod is `Ready`.
   - Applies configuration changes from `registry-node-config-<Node Name>` Secret to the registry-nodeservices static pod.
2. **External container registry**: The module forwards internal requests to the external container registry.

The following external components interact with the module:

1. **Containerd**: Sends requests to the registry to pull images used to create containers.
2. **Deckhouse-controller**: Sends requests to the registry to pull images used to install modules.
3. **Operator-trivy (as well as other components that access the registry directly)**: Sends requests to the registry to download images, for example, to perform security tests (in case of operator-trivy).

Additional resources

- Module `deckhouse` parameters
- Module `registry` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

DP cluster management

The `commander` and `commander-agent` modules are used to implement cluster management in Deckhouse Platform (DP).

A DP cluster with the `commander` module installed is a *managing* cluster. A DP cluster with the `commander-agent` module installed is a *managed* cluster.

Commander module

The `commander` module implements a web application that allows you to create standardized DP clusters and manage their configurations and lifecycle.

To install the `commander` module, you need a PostgreSQL instance. It can be deployed inside or outside the cluster by using the `managed-postgresql` module.

⚠ For `commander` in production environments, using a dedicated PostgreSQL instance is recommended.

When `commander` is installed, a PostgreSQL data encryption key is generated. This key is stored in the `d8-commander` namespace in the `commander-envs` Secret.

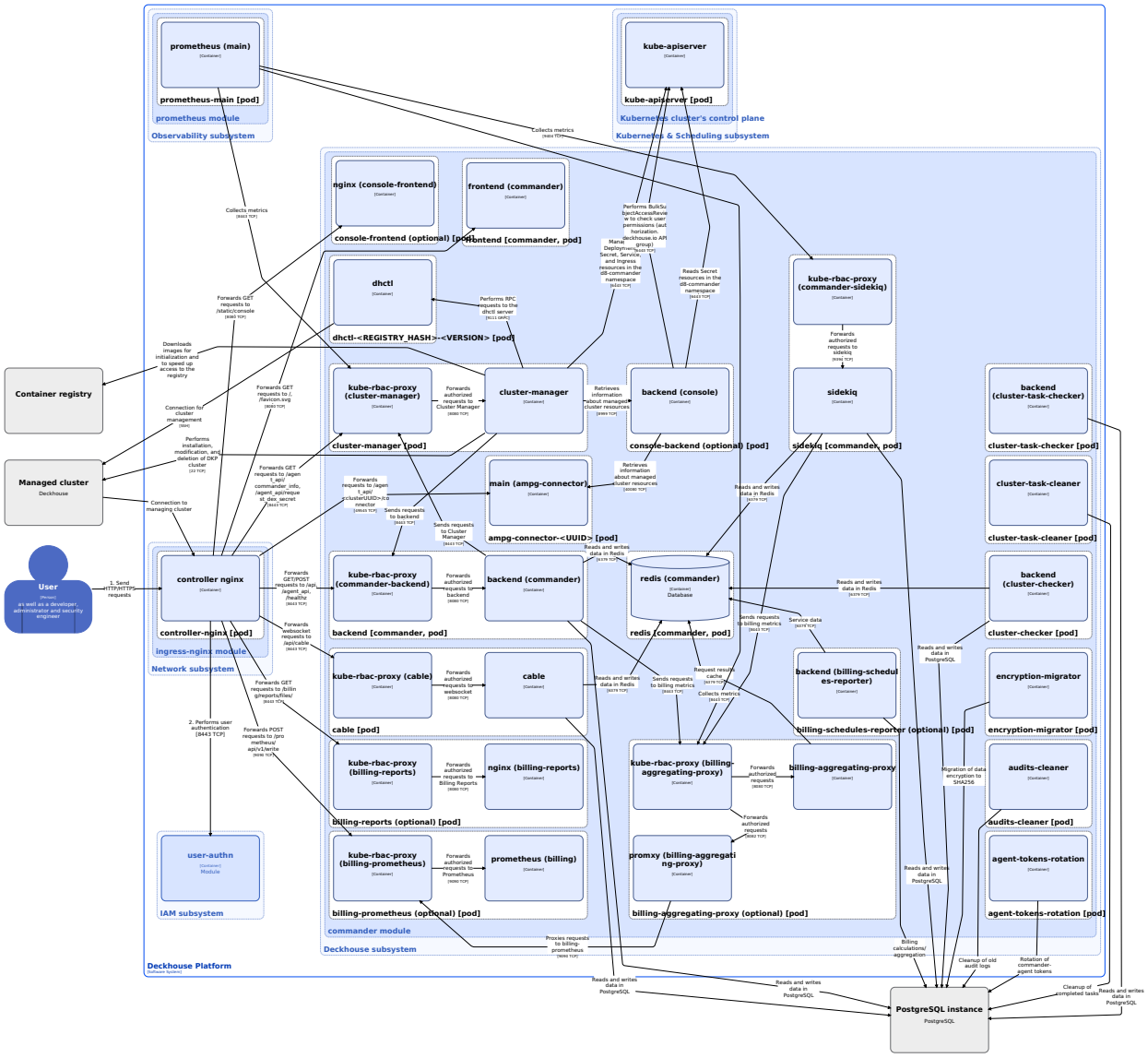
For details about features, configuration, and usage examples, see the module documentation section.

Commander module architecture

i The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.
- The diagram shows only the main containers of each component.

The Level 2 C4 architecture of the `commander` module and its interaction with other DP components are shown in the following diagram:



Commander module components

The module consists of core, service, and optional components.

Core components

Core components of the `commander` module include:

1. **Frontend** (Deployment): Provides a web interface for cluster management.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **frontend**: The main container.

2. **Backend** (Deployment): The main component that implements the following APIs:

   - UI API: Manages cluster, template, and workspace settings.
   - External API for integrations and automation: Operates on clusters, tasks, templates, projects, roles, and more.
   - API for cluster-manager: Exchanges cluster task states and processes deployment and configuration phases of managed clusters.
   - API for `commander-agent`: Authorizes and receives agent metrics, and provides target configuration for a managed cluster.

   Backend provides the following capabilities:

   - Cluster lifecycle management: Creates, modifies, or deletes a cluster (through cluster-manager), restarts failed tasks, and runs related checks.
   - Cluster RBAC management and API operation access restrictions.
   - Template management and storage of target and current cluster settings.
   - Container registry management.

- Project and project template management.
- Billing settings management (if this component is enabled).

The component stores the target system state in a PostgreSQL instance. It also uses a Redis instance to publish tasks to managed clusters and receive task processing results.

Contains the following containers:

- **wait-postgres**: An init container that checks PostgreSQL instance availability.
- **wait-migrations**: An init container that waits for all database migrations to complete.
- **backend**: The main container.
- **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

3. **Cable** (Deployment): Provides WebSocket support for real-time UI updates.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **cable**: The main container.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

4. **Sidekiq** (Deployment): A component based on the Sidekiq library that provides queue processing in Ruby on Rails.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **sidekiq**: The main container.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

5. **Cluster-manager** (Deployment): Executes tasks requested by backend to deploy managed DP clusters and manage their configuration.

   When a new cluster is created, cluster-manager creates the following resources in the `d8-commander` namespace of the managing cluster:

   - Deployment and Secret named `dhctl-<REGISTRY_HASH>-<VERSION>`: Runs dhctl in gRPC server mode. Cluster-manager uses RPC calls to install and configure a DP cluster.
   - To process incoming `commander-agent` connections and implement a reverse interaction channel with the control plane of the corresponding cluster:
     - Deployment (`ampg-connector-<UUID>`)
     - Service:
       - `ampg-connector-<UUID>`
       - `ampg-console-backend-<UUID>`
       - `ampg-api-server-<UUID>`
       - `ampg-deckhouse-tools-<UUID>`
       - `ampg-aggregating-proxy-<UUID>`
       - `ampg-upmeter-<UUID>`
       - `ampg-label-proxy-<UUID>`
     - Secret (`ampg-console-backend-<UUID>`)
     - Ingress (`ampg-agent-api-connector-<UUID>`, `ampg-console-backend-<UUID>`)
   - DexClient (`cluster-<UUID>`): Configures the authentication service for a managed cluster using the capabilities of the `user-authn` module.

   During cluster bootstrap, cluster-manager also performs the following actions in the managed cluster:

   - Installs the `commander-agent` module.
   - Disables the `terraform-manager` module.
   - Configures authentication based on the Dex controller of the `user-authn` module. As a Dex provider, cluster-manager sets `cluster-<UUID>` DexClient from the managing cluster.
   - Configures the managed cluster connection to a container registry.
   - If billing is enabled, creates a PrometheusRemoteWrite custom resource to send metrics to the billing-prometheus component in the managing cluster.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **cluster-manager**: The main container.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

6. **Dhctl-<REGISTRY_HASH>-<VERSION>** (Deployment): A component with a single **dhctl** container. It runs the dhctl utility in gRPC server mode. Cluster-manager uses RPC calls to the `dhctl` server to

install or configure a managed DP cluster.

7. **Ampg-connector-<UUID>** (Deployment): A component with a single **main** container that establishes a tunnel and proxies requests to the control plane of a managed DP cluster through the incoming `commander-agent` connection.

   The component is created by cluster-manager for each managed DP cluster.

8. **Cluster-checker** (Deployment): Periodically starts cluster verification tasks.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **backend**: The main container.

9. **Cluster-task-checker** (Deployment): Periodically searches for running tasks that have not reported their status for a long time and moves them to `lost` or `lost_in_critical`.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **backend**: The main container.

10. **Redis** (Deployment): A component with a single **redis** container that provides a dedicated Redis database instance for storing task queue data and commander session states.

11. **Console-frontend** (Deployment): Provides the managed DP cluster administration web interface.

    Contains the following containers:

    - **wait-postgres**: An init container that checks PostgreSQL instance availability.
    - **wait-migrations**: An init container that waits for all database migrations to complete.
    - **nginx**: The main container.

12. **Console-backend** (Deployment): An administration API backend for the managed DP cluster that serves requests from the cluster-manager component.

    Console-backend uses Secret and Service resources created by cluster-manager to connect to the control plane of a managed DP cluster.

    Contains the following containers:

    - **wait-postgres**: An init container that checks PostgreSQL instance availability.
    - **wait-migrations**: An init container that waits for all database migrations to complete.
    - **backend**: The main container.

Service components

Service components of the `commander` module include:

1. **Cluster-task-cleaner** (CronJob): Runs once a day and removes completed tasks created more than 30 days ago.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **cluster-task-cleaner**: The main container.

2. **Agent-tokens-rotation** (CronJob): Regularly rotates `commander-agent` tokens.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **agent-tokens-rotation**: The main container.

3. **Encryption-migrator** (CronJob): Runs once a day and migrates encrypted data from SHA-1 to SHA-256. Not all data may be processed in a single run. In addition, after restoring from backup or due to delayed migrations, SHA-1 records may appear.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **encryption-migrator**: The main container.

4. **Audits-cleaner** (CronJob): Runs once a day and deletes old audit logs.

The Deckhouse controller of the `deckhouse` module creates audits-cleaner if the `settings.featureFlags.auditsRetentionDays` module parameter is set and defines the audit log retention period.

Contains the following containers:

- **wait-postgres**: An init container that checks PostgreSQL instance availability.
- **wait-migrations**: An init container that waits for all database migrations to complete.
- **audits-cleaner**: The main container.

Optional components

Optional components of the `commander` module include:

1. **Billing-prometheus** (StatefulSet): Runs Deckhouse Prom++ in metrics ingestion mode over the Prometheus Remote Write protocol.

   The `prometheus` module in a managed DP cluster sends metrics. The component stores resource usage history for all managed clusters, which is an important data source for billing reports.

   Contains the following containers:

   - **prometheus**: The main container.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

2. **Billing-reports** (StatefulSet): A component that runs the nginx web server as billing report storage. Reports are uploaded to nginx over Web-based Distributed Authoring and Versioning (WebDAV).

   Contains the following containers:

   - **nginx**: The main container.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

3. **Billing-schedules-reporter** (Deployment): Implements a scheduler that starts and controls tasks for generating required billing reports.

   The billing-schedules-reporter component saves generated reports to billing-reports over WebDAV.

   Contains the following containers:

   - **wait-postgres**: An init container that checks PostgreSQL instance availability.
   - **wait-migrations**: An init container that waits for all database migrations to complete.
   - **backend**: The main container.

4. **Billing-aggregating-proxy** (Deployment): Aggregates PromQL queries, proxies them to billing-prometheus, and returns query results.

   Contains the following containers:

   - **billing-aggregating-proxy**: A sidecar container based on Grafana Mimir. It provides query optimization and data caching. If data is missing in the cache, Grafana Mimir forwards the request to billing-prometheus via the promxy sidecar container.
   - **promxy**: A sidecar container based on Promxy. It proxies requests to billing-prometheus and provides a single endpoint for accessing data from multiple Deckhouse Prom++ instances.
   - **kube-rbac-proxy**: A sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the main container.

> **i** The billing-prometheus, billing-reports, billing-schedules-reporter, and billing-aggregating-proxy components are created by the Deckhouse controller of the `deckhouse` module if `settings.featureFlags.billingEnabled` is set to `true`.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Works with Deployment, Service, Secret, and Ingress resources in the `d8-commander` namespace.
   - Authorizes requests.
2. **Container registry**: Retrieves available DP releases by release channels.
3. **PostgreSQL instance**: Stores cluster states and metadata, tasks, tokens, projects, billing parameters, and reports.

4. **Managed cluster**: Creates, modifies, or deletes DP clusters.

The following external components interact with the module:

- **Controller-nginx**: Forwards external requests to module component endpoints.
- **Prometheus-main**: Collects metrics from module components.

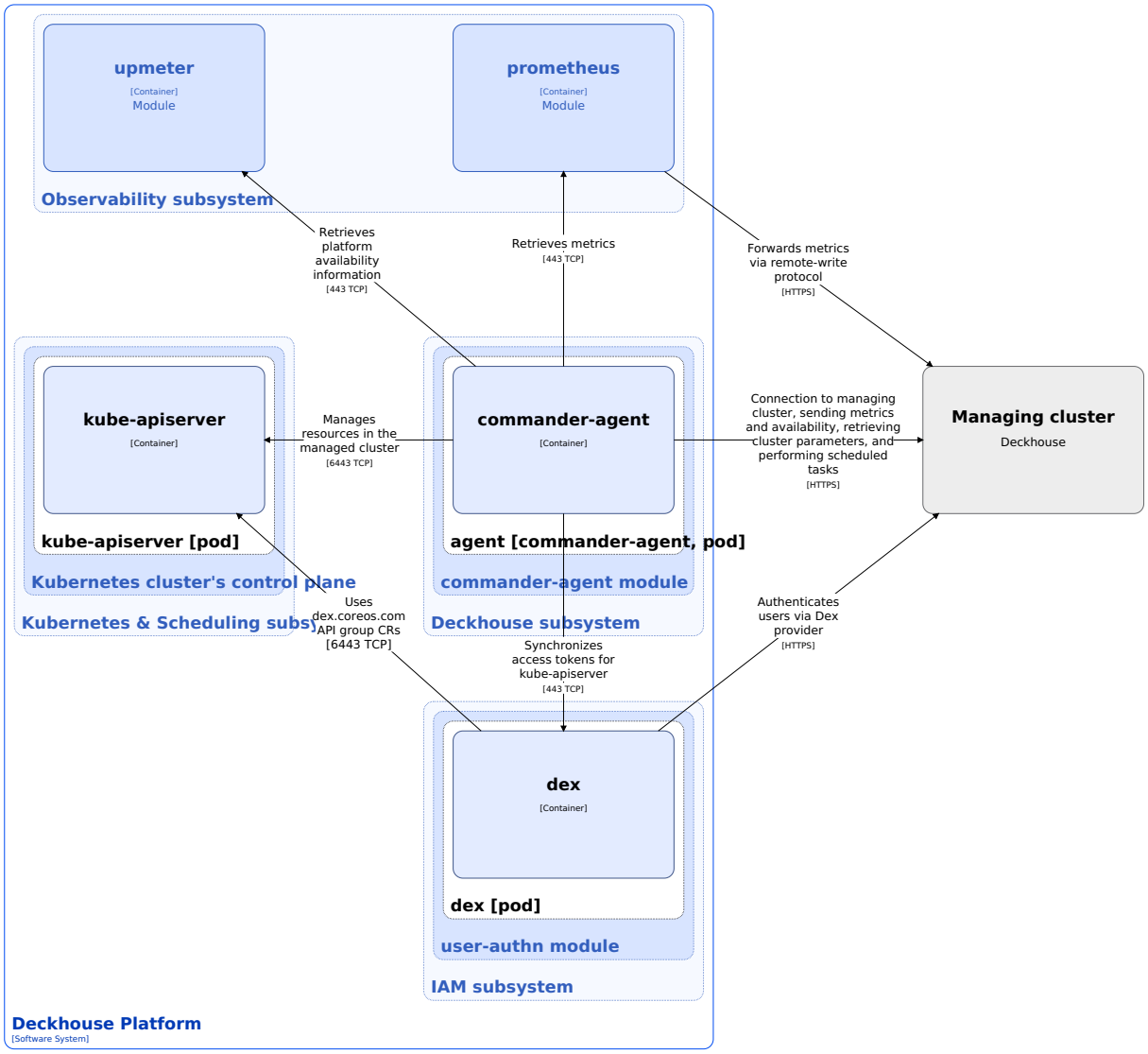
Commander-agent module

The `commander-agent` module establishes a service connection to a managing DP cluster.

For details about module settings, see the module documentation section.

Commander-agent module architecture

The Level 2 C4 architecture of the `commander-agent` module and its interaction with other DP components are shown in the following diagram:



Commander-agent module components

The module consists of one **agent** component and one **commander-agent** container.

Agent performs the following actions:

- Brings managed cluster configuration in line with target configuration received from `commander` module backend.
- Collects managed cluster metrics:
  - `CPU cores`: Number of node CPU cores.
  - `Memory`: Total node memory in GiB.
  - `Filesystem`: Total size of node root disks in GiB.
  - `PVC`: Total size of attached disks in GiB.
  - `Nodes`: Total number of nodes.

- Collects cluster availability information.
- Establishes a secure connection to the managing DP cluster.
- Sends metrics and cluster availability information to the managing DP cluster.
- Configures user authentication via the Dex provider of the managing cluster.
- If billing is enabled in the managing cluster, creates a PrometheusRemoteWrite custom resource to send metrics to the billing service of the managing DP cluster.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**: Manages resources of the managed cluster.
2. **Prometheus**: Retrieves metrics from the managed cluster.
3. **Upmeter**: Retrieves availability information of the managed cluster.
4. **Dex**: Synchronizes access tokens for Kubernetes API requests from the managing cluster to the managed cluster.
5. **Managing cluster**:
   - Organizes a proxy connection to the managing cluster.
   - Receives target cluster configuration.
   - Sends metrics to the billing service.

Additional resources

- Module `commander` documentation
- Module `commander-agent` documentation
- Module `managed-postgresql` documentation
- Module `prometheus` documentation
- Module `terraform-manager` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Delivery subsystem

This section describes the architecture of the Delivery subsystem in Deckhouse Platform (DP).

The Delivery subsystem includes the following modules:

- `operator-argo`: Manages Argo CD installations in DP.
- `operator-helm`: Declaratively manages Helm chart deployments.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Operator-argo module

The `operator-argo` module deploys Argo CD Operator in a Deckhouse Platform (DP) cluster. The module enables to install Argo CD in a DP cluster using the ArgoCD resource.

The module works with the following custom resources:

- Application: Describes application deployment and its management.
- ApplicationSet: Provides templating and mass application creation according to defined rules.
- AppProject: Defines a set of applications and application access policies.
- ArgoCD: Main resource for deploying and configuring an Argo CD instance.
- ArgoCDExport: Exports Argo CD configuration and state for backup or migration.
- ImageUpdater: Automatically updates application container images.
- NamespaceManagement: Defines namespace management rules for an Argo CD instance.

- NotificationsConfiguration: Defines notification settings for events in Argo CD and applications.

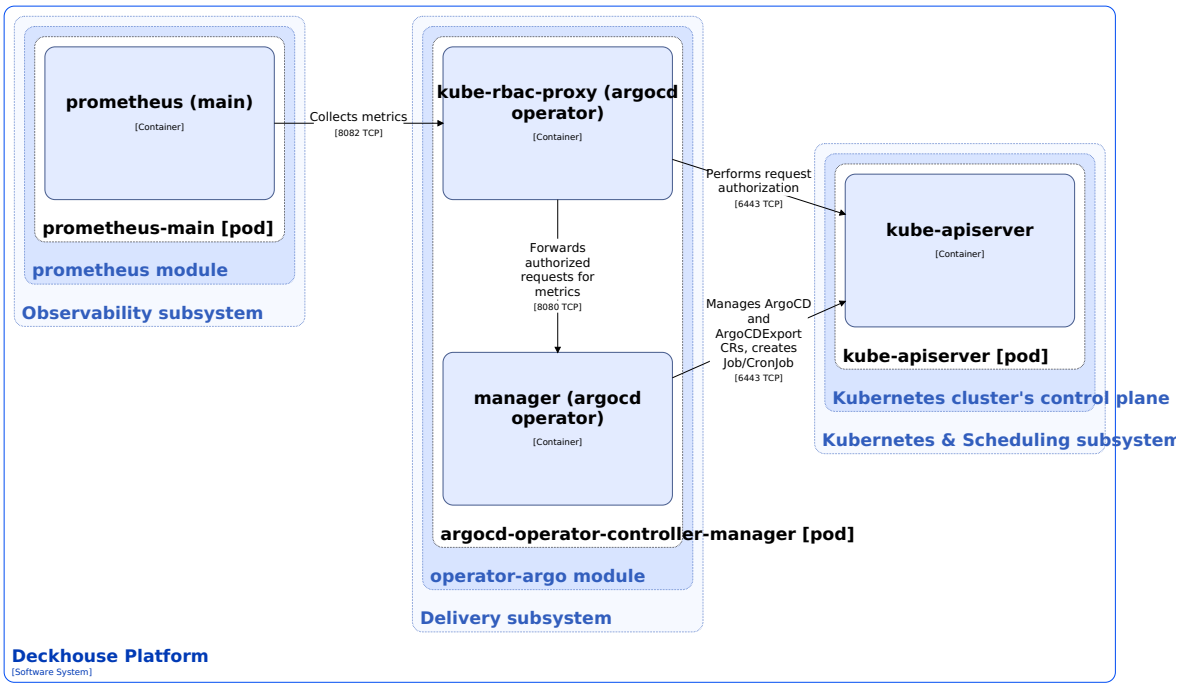
For more details on module settings and usage examples, refer to the module documentation.

Module architecture

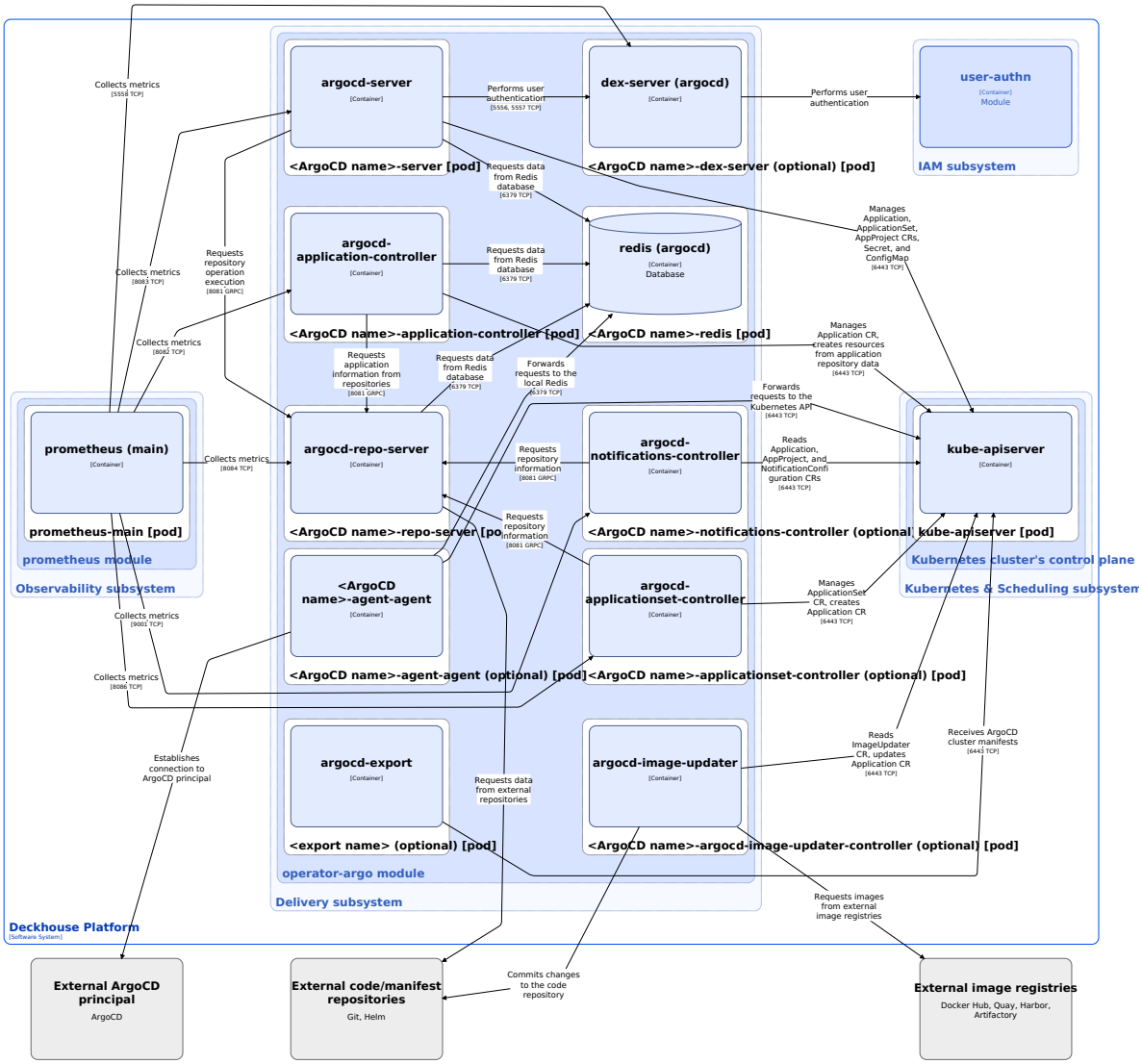
> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.
> - Only the main containers of each component are shown in the diagram.

The Level 2 C4 architecture of the `operator-argo` module and its interactions with other DP components are shown in the following diagrams:

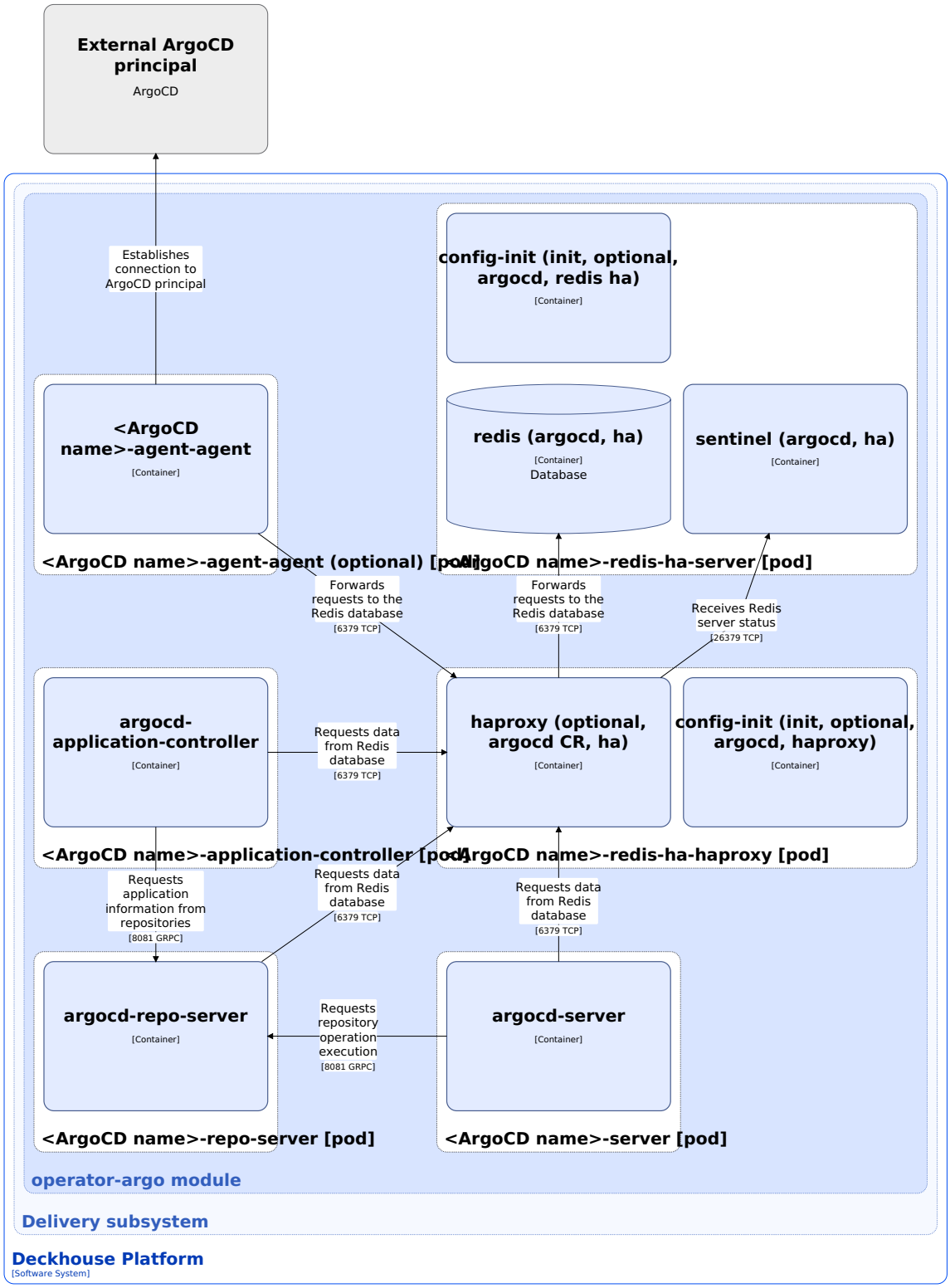
- Main module operator:



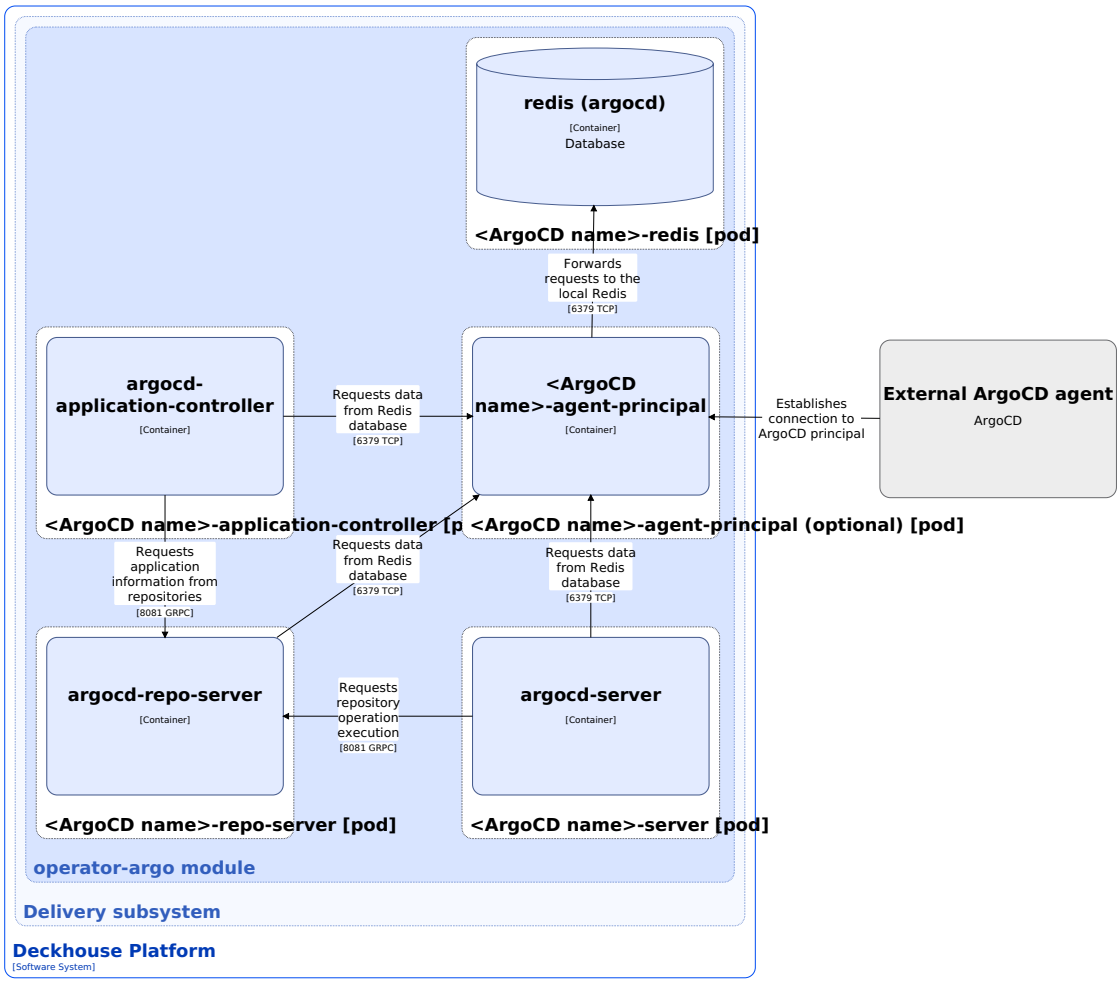
- Argo CD instance deployment scenario with Redis in a non-HA configuration:



- Argo CD instance deployment scenario with Redis in an HA configuration (the diagram shows only differences from the primary deployment scenario):



- Argo CD instance deployment scenario in a principal cluster for a multicluster setup (the diagram shows only differences from the primary deployment scenario):



Module components

The module consists of the following components:

1. **Argocd-operator-controller-manager** (Deployment): Implementation of Argo CD Operator that allows deploying Argo CD instances in a DP cluster. The component works with the following custom resources:
   - ArgoCD: Main resource for deploying and configuring an Argo CD instance.
   - ArgoCDExport: Exports Argo CD configuration and state for backup or migration. The operator reads the ArgoCDExport custom resource and creates a Job/CronJob with the same name as the ArgoCDExport resource. The created Job/CronJob performs backup of the Argo CD instance configuration.
   - NamespaceManagement: Defines namespace management rules for an Argo CD instance. The operator watches the NamespaceManagement custom resource and updates the `argocd-cmd-params-cm` ConfigMap accordingly.
   - NotificationsConfiguration: Defines notification settings for events in Argo CD and applications. The operator reads NotificationsConfiguration custom resources and updates configuration in the `argocd-notifications-cm` ConfigMap based on them.

   Argocd-operator-controller-manager creates Deployment, Secret, ConfigMap, StatefulSet, and other resources for each ArgoCD custom resource, adding that resource name as a prefix to the created resources.

   It consists of the following containers:

   - **manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to `manager` metrics.

   > **i** The following components describe resources created by argocd-operator-controller-manager based on configuration defined in the ArgoCD custom resource. The `<ArgoCD name>` prefix is used in descriptions and is replaced by the controller with the ArgoCD resource name.

2. **<ArgoCD name>-server** (Deployment): Argocd-server. Main component for interacting with an Argo

CD instance. Argocd-server provides REST/gRPC API and a web UI for Argo CD management. The component allows managing Application, ApplicationSet, and AppProject custom resources through the provided interfaces (web UI, API, CLI).

The operator creates this component if the `.spec.server.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

It consists of the following containers:

- **argocd-server-init**: Optional set of init containers defined by the user in the `.spec.server.initContainers` parameter of the ArgoCD custom resource.
- **rollout-extension**: Optional init container that loads the UI extension for the Rollout custom resource. The module does not provide a controller for this custom resource. Such controller must be installed and configured separately. Argocd-operator-controller-manager adds rollout-extension if `.spec.server.enableRolloutsUI` is set to `true`.
- **argocd-server**: Main container.
- **argocd-server-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.server.sidecarContainers` parameter of the ArgoCD custom resource.

3. **<ArgoCD name>-repo-server** (Deployment): Argocd-repo-server. Component responsible for template rendering, application manifest generation, and working with external repositories used by Argo CD. Argocd-repo-server synchronizes application manifests from configured repositories and passes them to the corresponding components for further deployment.

   The operator creates this component if the `.spec.repo.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   It consists of the following containers:

   - **copyutil**: Init container that copies executables for use by the main container.
   - **argocd-repo-server-init**: Optional set of init containers configured through the `.spec.repo.initContainers` parameter of the ArgoCD custom resource to prepare the environment.
   - **argocd-repo-server**: Main container that generates and processes manifests and works with remote application Git repositories.
   - **argocd-repo-server-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.repo.sidecarContainers` parameter of the ArgoCD custom resource and used to extend repo-server functionality.

4. **<ArgoCD name>-application-controller** (StatefulSet): Argocd-application-controller. Component responsible for synchronization and state management of applications defined in Argo CD. Argocd-application-controller provides idempotent application of Kubernetes manifests, manages deployment, rollback, and self-healing workflows, and monitors resource state in the cluster.

   The operator creates this component if the `.spec.controller.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   It consists of the following containers:

   - **application-controller-init**: Optional set of init containers configured through the `.spec.controller.initContainers` parameter of the ArgoCD custom resource to prepare the environment.
   - **argocd-application-controller**: Main container implementing synchronization logic for Application custom resources and resources created from them.
   - **application-controller-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.controller.sidecarContainers` parameter of the ArgoCD custom resource and used to extend controller capabilities.

5. **<ArgoCD name>-applicationset-controller** (Deployment): Argocd-applicationset-controller. Optional component consisting of a single **applicationset-controller** container and responsible for managing the ApplicationSet custom resource in Argo CD. It allows automatic creation, update, and deletion of Application resources based on configured templates and generators (for example, Git, List, Matrix, and Cluster generators). This simplifies mass management of similar applications that must be deployed across different environments or clusters.

   The operator creates this component if the `.spec.applicationSet.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   For more details about the component, refer to the applicationset-controller documentation.

6. **<ArgoCD name>-argocd-image-updater-controller** (Deployment): Argocd-image-updater-controller. Optional component consisting of a single **argocd-image-updater** container and intended for automatic container image updates in Argo CD applications when new versions appear in image registries. The component tracks image tag changes and, when a new version is found, updates corresponding Application resources in Argo CD (for example, image tags in manifests or Helm values) via a pull request to a Git repository or directly, depending on the selected workflow.

   Argocd-image-updater-controller performs the following functions:

- Manages the ImageUpdater custom resource that defines settings for automatic updates of application container images.
- Periodically checks application container images in supported registries (Docker Hub, Quay.io, Harbor, and others).
- Supports filtering image tags by patterns and update strategies (`semver`, `latest`, and others).
- When a new image version is found, automatically performs write-back (writes the new image tag value) to Argo CD Application or a Git repository depending on the configured method.

For correct operation, the component requires access to Git repositories and, if needed, private image registries. Credentials for accessing image registries can be stored in Kubernetes Secrets.

To enable the component, set `.spec.imageUpdater.enabled` to `true` in the ArgoCD custom resource.

For more details about the component, refer to the argocd-image-updater documentation.

7. **<ArgoCD name>-notifications-controller** (Deployment): Optional controller consisting of a single **argocd-notifications-controller** container that sends notifications about Argo CD events (for example, successful application sync, deployment failures, status changes, and others) to external notification systems, including email, Slack, Microsoft Teams, Telegram, OpsGenie, Webhook, and others.

   The main module operator argocd-operator-controller-manager generates notification settings based on NotificationsConfiguration custom resources and stores them in the `argocd-notifications-cm` ConfigMap and `argocd-notifications-secret` Secret used by the controller to generate and send notifications.

   To enable the component, set `.spec.notifications.enabled` to `true` in the ArgoCD custom resource.

   For more details on operation, refer to the Argo CD Notifications documentation.

8. **<ArgoCD name>-dex-server** (Deployment): Argocd-dex-server. Optional component for user authentication in Argo CD, acting as an OIDC provider (OpenID Connect) based on Dex. The component enables user login through various external authentication providers (LDAP, GitHub, GitLab, SAML, Azure AD, and others) and supports static users defined in Dex configuration.

   It consists of the following containers:

   - **copyutil**: Init container that copies executables for use by the main container.
   - **dex**: Main container.

   To enable the component, define parameters in `.spec.sso.dex` section of the ArgoCD custom resource.

   > ⚠ For Argo CD user authentication in DP, the `operator-argo` module supports integration with the `user-authn` module (built-in DP authentication). Other external providers via Dex are not used in this configuration.
   >
   > For more details about module usage examples, refer to the corresponding documentation section.

9. **<ArgoCD name>-redis** (Deployment): Argocd-regis. Mandatory component consisting of a single **redis** container and responsible for storing task queue data and session state in Argo CD. Argocd-regis provides a dedicated Redis database instance.

   Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `false`.

10. **<ArgoCD name>-redis-ha-server** (StatefulSet): Argocd-redis-ha-server. Mandatory component for deploying Redis in high availability (HA) mode in Argo CD. It provides a fault-tolerant Redis cluster with replication and automatic failover using Redis Sentinel.

    It consists of the following containers:

    - **config-init**: Init container that prepares configuration for Redis and Sentinel before main containers start.
    - **redis**: Main container implementing a Redis server instance.
    - **sentinel**: Auxiliary container running Redis Sentinel to monitor Redis instance health and automatically switch to a replica if the primary instance fails.

    Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `true`.

11. **<ArgoCD name>-redis-ha-haproxy** (Deployment): Argocd-redis-ha-proxy. Additional component for load balancing and distributing traffic to Redis cluster instances (redis-ha-server) based on the HAProxy load balancer/reverse proxy server.

It consists of the following containers:

- **config-init**: Init container that prepares HAProxy configuration before the main container starts.
- **haproxy**: Container acting as a proxy server and providing transparent routing of client requests to available Redis master/replica instances, as well as automatic switching between them on failover.

Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `true`.

> **i**
>
> The following components are used to implement a secure and scalable multicluster setup in Argo CD:
>
> - Principal: The central control point. It stores state and distributes tasks.
> - Agent: The executor in each target cluster. It applies manifests and reports back.
>
> This approach allows you to manage multiple clusters without direct access from the central Argo CD to the Kubernetes API of each cluster, which reduces the number of required open inbound connections, provides isolation, and ensures fault tolerance.

12. **<ArgoCD name>-agent-agent** (Deployment): Argocd-agent-agent. Optional component consisting of a single **<ArgoCD name>-agent-agent** container and responsible for executing operations on managed Kubernetes cluster resources based on requests from Argo CD. The component establishes connection to Argo CD Principal, synchronizes applications, and manages their state based on commands received from Argo CD Principal.

    For details on Argo CD multicluster architecture, refer to the Argo CD documentation.

    Argocd-operator-controller-manager deploys this component if `.spec.argoCDAgent.agent.enabled` in the ArgoCD custom resource is `true`. Argo CD Agent and Argo CD Principal cannot be enabled simultaneously in a single ArgoCD resource.

13. **<ArgoCD name>-agent-principal** (Deployment): Argocd-agent-principal. Optional component consisting of a single **<ArgoCD name>-agent-principal** container and enabling Argo CD operation in a multicluster setup.

    When this component is enabled, argocd-operator-controller-manager reconfigures all components that use Redis connection to use Redis proxy instead. The Argocd-agent-principal component provides this Redis proxy and routes database requests by analyzing Redis keys: depending on key values, a request is sent either to a local Redis instance or to one of remote Argo CD Agents.

    Argocd-operator-controller-manager deploys this component if `.spec.argoCDAgent.principal.enabled` in the ArgoCD custom resource is `true`. Argo CD Agent and Argo CD Principal cannot be enabled simultaneously in a single ArgoCD resource.

14. **<Export name>** (Job/CronJob): Argocd-export. Optional component implemented as Job or CronJob that creates a pod with a single **argocd-export** container. The component creates backup of Argo CD instance configuration and state.

Module interactions

The module interacts with the following components:

1. **External image registries**: Receives image lists.
2. **External code repositories**:
   - Receives application deployment manifests from repositories.
   - Updates `image` in Helm chart source code.
3. **External Argo CD Principal**:
   - Connects to Argo CD principal cluster.
   - Receives processing requests.
   - Sends back processing results.
4. **Kube-apiserver**:
   - Manages Application, ApplicationSet, AppProject, ArgoCD, ArgoCDExport, ImageUpdater, NamespaceManagement, NotificationsConfiguration custom resources, as well as Secret and ConfigMap.
   - Manages resources created during deployment of user applications described in the Application custom resource.
   - Authorizes requests for metrics retrieval.
5. **`user-authn` module**: Redirects user for authentication.

The following external components interact with the module:

1. **Prometheus-main**: Collects metrics provided by the operator and Argo CD instances.
2. **External Argo CD Agent**:
   - Connects to Argo CD principal cluster.
   - Receives processing requests.

- Sends back processing results.

Additional resources

- Module `operator-argo` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Kubernetes & Scheduling subsystem

This subsection describes the architecture of the modules that are part of the Kubernetes & Scheduling subsystem of Deckhouse Platform (DP).

The Kubernetes & Scheduling subsystem includes the following modules:

- `control-plane-manager`: Main module of the subsystem, responsible for managing cluster control plane components.
- `descheduler`: Analyzes the cluster state and evicts pods in accordance with the active strategies.
- `pod-reloader`: Allows you to automatically restart the workload if certain ConfigMap or Secret changes.
- `vertical-pod-autoscaler`: Automatically adjusts container resource requests and limits in pods based on actual consumption. The module architecture is described on the corresponding page.
- `adaptive-resource-management`: Automatically selects optimal resource requests and limits for containers using VPA (Vertical Pod Autoscaler) recommendations.

This subsection also describes the architecture of the control plane and the kubelet agent.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster control plane

Deckhouse Platform (DP) uses a standard (“vanilla”) Kubernetes cluster. The cluster control plane includes the following core components:

1. **kube-apiserver**: Kubernetes API server. It processes REST requests, provides access to the overall cluster state through which all other components interact, validates Kubernetes API resources, and stores them in **etcd** storage. It includes the following containers:
   - **kube-apiserver**: Main container.
   - **kube-apiserver-healthcheck**: Sidecar container that enables health checks of **kube-apiserver** without enabling anonymous authentication and without exposing an unauthenticated port. It uses a client certificate to authenticate to the API server. It is an open source project.
2. **etcd**: Distributed key-value storage that holds all configuration data and Kubernetes cluster resources.
3. **kube-scheduler**: Kubernetes scheduler. It analyzes node resources and assigns pods based on constraints and rules such as affinity and taints.
4. **kube-controller-manager**: Kubernetes controller manager. It runs controller loops that monitor and reconcile the state of standard Kubernetes resources to match the desired state. Examples of controllers shipped with Kubernetes: replication controller, endpoints controller, namespace controller, and ServiceAccount controller.

Interactions between the Kubernetes control plane components are shown in the architecture diagram of the `control-plane-manager` module.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Kubelet

Kubelet is not a control plane component, but it plays a key role in the operation of a Kubernetes cluster.

Kubelet is an agent that runs on every node in a Kubernetes cluster. It ensures that containers in pods are started and run according to their specifications. Kubelet continuously interacts with kube-apiserver to verify and maintain the state of nodes and containers. It is also responsible for starting control plane components.

Static pod manifests

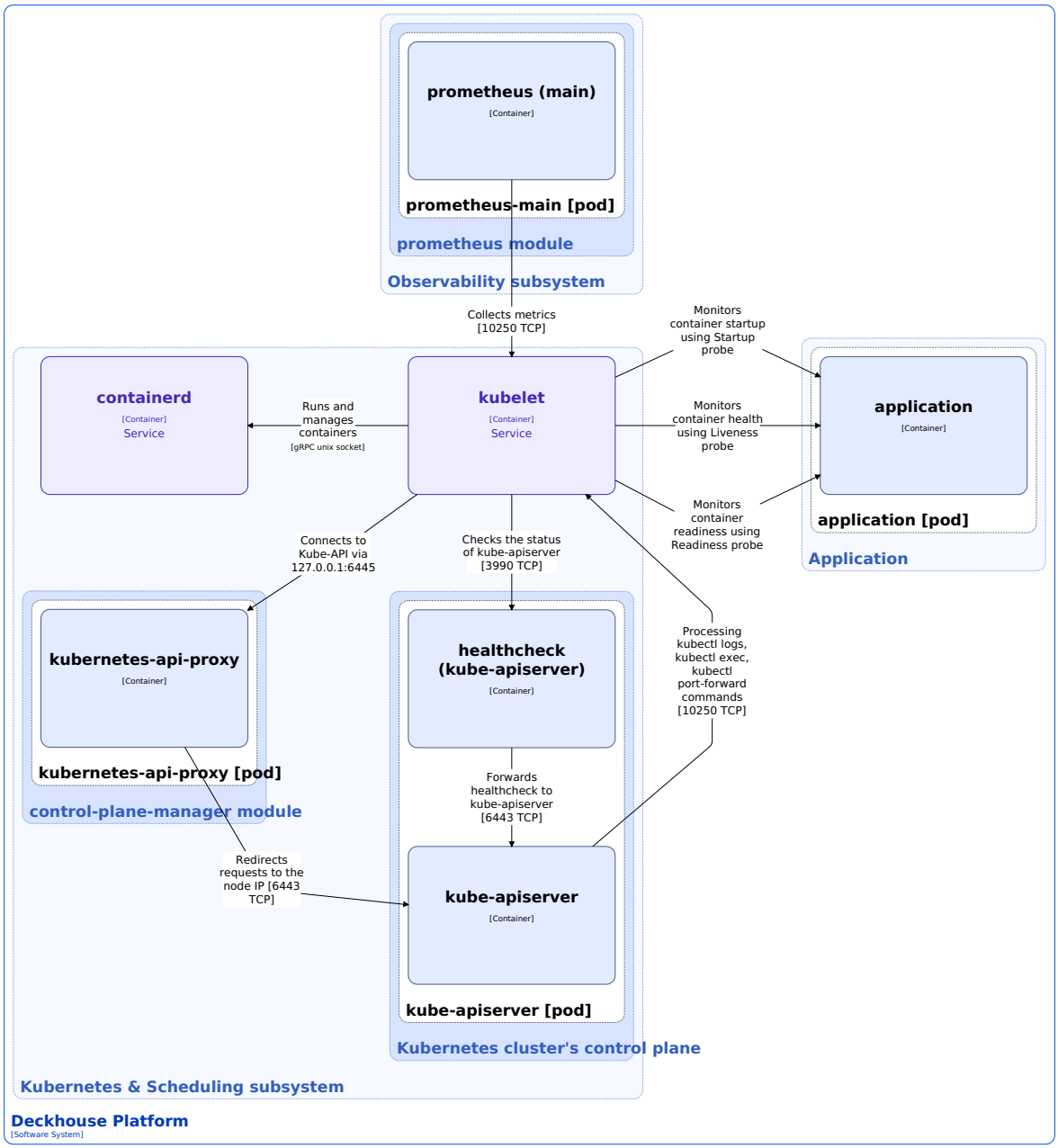
Kubelet starts control plane components from static pod manifests located in the `/etc/kubernetes/manifests` directory. In Deckhouse Platform, kubelet processes only files with the `.yaml` or `.yml` extension in this directory.

Files with other extensions, such as `kube-apiserver.backup`, `kube-apiserver.yaml.bak`, editor swap files, or other temporary files, are ignored. This prevents accidental processing of backup or non-manifest files as static pod manifests.

Kubelet interactions

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.
> - The diagram shows an `application` pod, which represents any pod in the cluster, both system and user-managed.

Kubelet interactions are shown in the following diagram:



Kubelet monitors the state of containers in all pods running on the node, including both user workloads and DP components, by performing Startup, Liveness, and Readiness probes according to the pod specification. For more information about probes, see the Kubernetes documentation.

Kubelet also interacts with the following components:

1. **Containerd**: Receives commands from kubelet to manage the container lifecycle on the node via the Container Runtime Interface (CRI).
2. **Kubernetes-api-proxy**: Proxies requests to kube-apiserver that are sent to the `localhost` address. It is part of the `control-plane-manager` module.
3. **Kube-apiserver-healthcheck**: Checks the health of kube-apiserver.

The following components interact with kubelet:

1. **Kube-apiserver**:
   - Retrieving pod logs (the `kubectl logs` command).
   - Executing commands in running pods (the `kubectl exec` command).
   - Port forwarding (the `kubectl port-forward` command).
2. **Prometheus-main**: Collects kubelet metrics.

Additional resources

- Module `control-plane-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster control plane management

Control-plane-manager module

Cluster control plane components are managed by the `control-plane-manager` module, which runs on all master nodes (nodes labeled with `node-role.kubernetes.io/control-plane: ""`).

The module operates with the following custom resources:

- ControlPlaneNode: Describes the parameters and state of control plane nodes (master nodes) in the cluster. It's used for managing the lifecycle and configuration of each control plane component.
- ControlPlaneOperation: Defines operations on control plane components (upgrade, downgrade, addition, or removal of components), and allows tracking and managing the execution of these operations at the cluster level.
- KubeSchedulerWebhookConfiguration: Describes the parameters and logic for connecting external webhooks to the `kube-scheduler` component to extend its functionality.

> ⚠ The ControlPlaneNode and ControlPlaneOperation custom resources are available to users in read-only mode. Full lifecycle management of these resources is performed exclusively by the `control-plane-manager` module.

Control plane management functions:

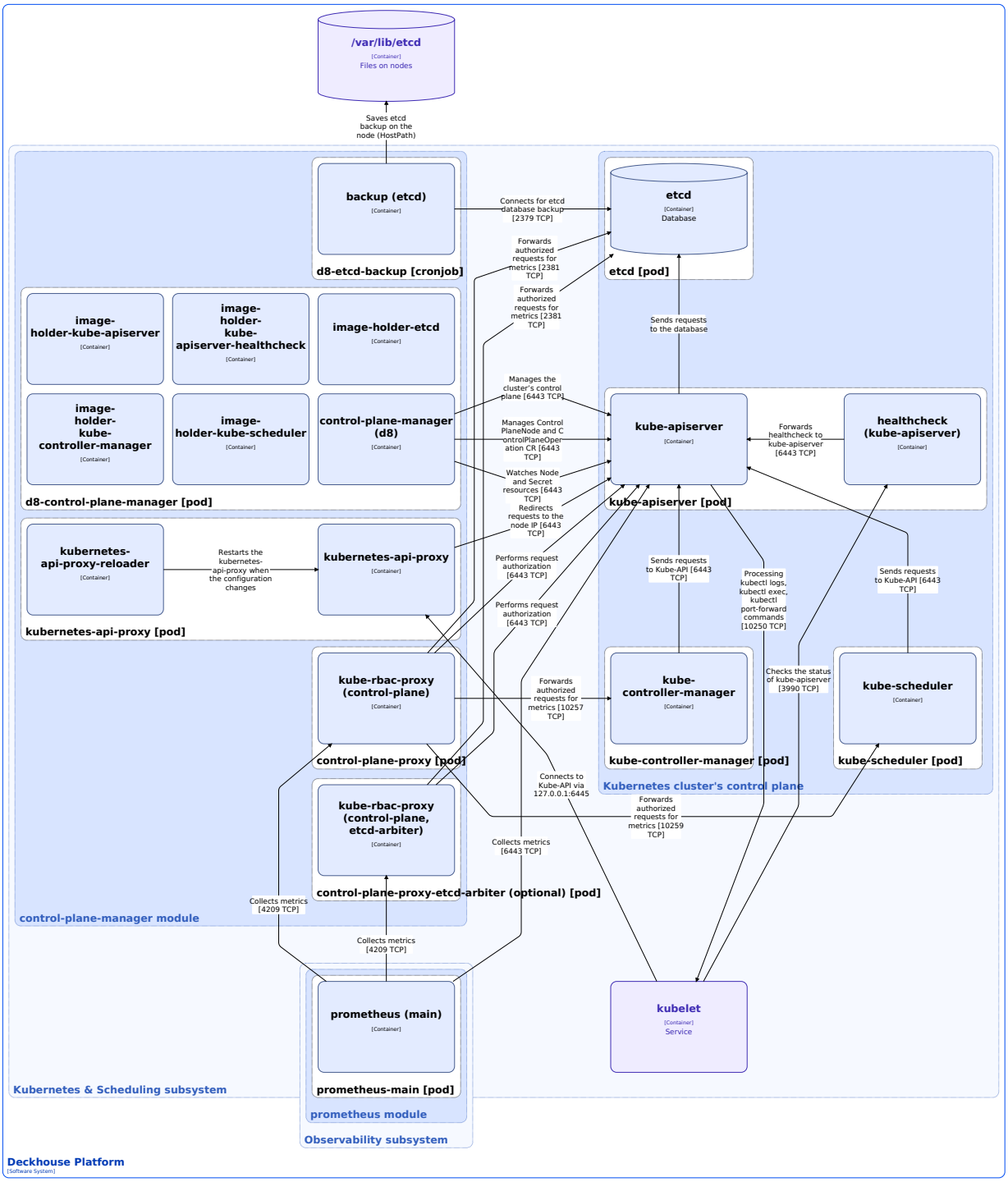
- **Certificate management**: Issuing, renewing, and rotating certificates required for control plane operation. Ensures automatic and secure control plane configuration and allows additional Subject Alternative Names (SAN) to be added for secure access to the Kubernetes API.
- **Component configuration**: Automatic generation of the required configuration files and manifests for control plane components.
- **Component upgrade and downgrade**: Maintains consistent component versions across the cluster.
- **Management of etcd cluster configuration**: Scaling master nodes and migrating between single-master and multi-master configurations.
- **Management of kubeconfig**: Maintains an up-to-date configuration for using `kubectl` on cluster nodes. Generates, renews, and updates the kubeconfig with *cluster-admin* privileges and creates a symbolic link for the `root` user so that the kubeconfig is used by default.
- **Scheduler extension**: Enables external plugins via webhooks using the KubeSchedulerWebhookConfiguration resource. This allows advanced scheduling logic when planning cluster loads, for example:
  - Placing data-intensive application pods closer to their data.
  - Prioritizing nodes based on their state (network load, storage subsystem health, etc.).
  - Dividing nodes into zones, etc.
- **Periodic etcd defragmentation**. In clusters with three or more etcd members, this feature is enabled by default. For more details, see the `control-plane-manager` module documentation.

For detailed configuration options and usage examples, refer to the `control-plane-manager` module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `control-plane-manager` module and its interactions with other platform components are shown in the following diagram:



Module components

The module consists of the following components:

1. **d8-control-plane-manager** (DaemonSet): Manages cluster control plane components and runs on all master nodes.

   The **d8-control-plane-manager** controller performs the following actions:

   - Monitors the `d8-control-plane-manager-config` and `d8-pki` Secrets and, based on their information, creates or updates the ControlPlaneNode custom resource for each master node.
   - If the required node configuration differs from the current one, creates a ControlPlaneOperation resource to perform operations to update the node's configuration.
   - Determines the order in which to execute the requested ControlPlaneOperation operations to maintain the required cluster fault tolerance during updates.
   - Monitors the execution of operations specified in the ControlPlaneOperation resource.
   - After the requested operations are completed, updates the current state of the master node in the ControlPlaneNode resource.

It consists of the following containers:

- **control-plane-manager**: Main container. Developed by Flant.
- A set of sidecar containers used to pre-pull images of control plane components. These containers remain paused and serve only as image holders:
  - **image-holder-kube-apiserver**
  - **image-holder-kube-apiserver-healthcheck**
  - **image-holder-kube-controller-manager**
  - **image-holder-kube-scheduler**
  - **image-holder-etcd**

2. **kubernetes-api-proxy** (static pods): Additional proxy server configured on each master node to handle requests to `localhost`. By default, it proxies requests to the local **kube-apiserver** instance. If the latter is unavailable, it sequentially queries the remaining **kube-apiserver** instances. It includes the following containers:
   - **kubernetes-api-proxy**: NGINX-based proxy server.
   - **kubernetes-api-proxy-reloader**: Sidecar container that restarts the proxy server when its configuration changes. Developed by Flant.
3. **control-plane-proxy** (DaemonSet): Component that is installed in the cluster when the [`prometheus`] module(/modules/prometheus/) is enabled. In this case, the control-plane-proxy runs on all master nodes and forwards authorized requests for metrics of the following components of the cluster control plane:
   - **kube-controller-manager**;
   - **kube-scheduler**;
   - **etcd**.

   It consists of a single container:
   - **kube-rbac-proxy***: Container with an authorization proxy based on Kubernetes RBAC, providing secure access for metrics of the components of the cluster control plane. It is an open source project.
4. **control-plane-proxy-etcd-arbiter** (DaemonSet): Optional component that is installed in the cluster when the `prometheus` module is enabled, if the cluster operates in HA mode with two master nodes and an arbiter node. In this case, Control-plane-proxy-etcd-arbiter runs on the arbiter node and forwards authorized requests for metrics to the etcd instance running on it.

   It consists of a single container:
   - **kube-rbac-proxy***: Container with an authorization proxy based on Kubernetes RBAC, providing secure access for metrics of the etcd instance (described above).
5. **d8-etcd-backup** (CronJob): Periodically performs backups of the cluster's **etcd** database. It consists of the following container:
   - **backup**: Container running a shell script that creates an etcd snapshot using `etcdctl` and stores it in `/var/lib/etcd` on the master node (default directory, configurable via the module parameters).

Module interactions

The module interacts with the following components:

1. **kube-apiserver**:
   - Manages cluster control plane components.
   - Reconciles ControlPlaneNode and ControlPlaneOperation custom resources.
   - Watches `d8-control-plane-manager-config` and `d8-pki` Secrets.
   - Proxies and load-balances requests to **kube-apiserver** sent to `localhost`.
   - Authorizes the requests for metrics.
2. **etcd**:
   - Manages etcd cluster configuration and membership.
   - Performs periodic database backups.

The following external components interact with the module:

- **kubelet**: Requests to **kube-apiserver** sent to `localhost` are proxied by the module's **kubernetes-api-proxy** component.
- **Prometheus-main**: Collects metrics of the control plane components.

Cluster control plane monitoring

The `control-plane-manager` module provides control plane monitoring, ensuring secure metrics collection and providing a basic set of monitoring rules for the following cluster components:

- **kube-apiserver**
- **kube-controller-manager**
- **kube-scheduler**
- **etcd**

Metrics from **kube-apiserver** are collected directly by **prometheus-main**. Metrics of the other control-plane components are collected by **prometheus-main** with authorization in kube-apiserver via the control-plane-proxy component. The `control-plane-manager` module adds the corresponding metric collection rules to the **prometheus-main** configuration.

Additional resources

- Module `control-plane-manager` documentation
- Module `prometheus` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Vertical Pod Autoscaler

The `vertical-pod-autoscaler` module provides Vertical Pod Autoscaler (VPA) in Deckhouse Platform (DP).

For details on module configuration and usage examples, refer to the relevant documentation section.

VPA operating modes

VPA can operate in two modes:

- Automatic resource adjustment:
  - **InPlaceOrRecreate** (the default, available in Kubernetes 1.33 and later): VPA attempts to update resources without recreating Pods. If in-place resource updates are not possible, VPA falls back to behavior similar to the **Recreate** mode: the Pod for which the resources cannot be updated is evicted, and the controller creates a new Pod with updated resources.

    > To use the **InPlaceOrRecreate** mode in Kubernetes versions earlier than 1.33, enable the `InPlacePodVerticalScaling` feature gate in the `control-plane-manager` configuration.

  - **Recreate**: VPA adjusts the resources of running Pods by restarting them. For a single Pod (`replicas: 1`), this will result in service unavailability during the restart. VPA does not restart Pods that were created without a controller.
  - **Auto**: A deprecated mode. It will no longer be supported in future DP versions. Use one of the supported operating modes instead.
- Recommendations only, without modifying resources:
  - **Initial**: Pod resources are adjusted only when Pods are created and not during the runtime.
  - **Off**: VPA does not change resources automatically. However, it still provides resource recommendations, which can be viewed using the `d8 k describe vpa` command.

When VPA is enabled and configured, resource requests are set automatically based on Prometheus data. You can also configure the system to only provide recommendations without applying any changes. For details on enabling and configuring the VPA, refer to Administration.

VPA limitations

Before using the vertical pod autoscaler (VPA), you need to consider several limitations:

- Pod restarts when resources are changed:
  - When using modes that allow Pod recreation, VPA may recreate a Pod if requested resources cannot be updated in place.
  - The new Pod can be scheduled on a different node.
- Compatibility with Horizontal Pod Autoscaler (HPA):
  - It is not recommended to use VPA together with HPA that is set for scaling based on CPU and

    memory.
  - VPA can be used with HPA that is set for scaling based on custom or external metrics.

- Issues in large clusters: VPA can work in large clusters, but the load on VPA increases as the number of pods grows.
- Issues with Pending pods: VPA may recommend resources that exceed those available in the cluster, causing pods to get stuck in the `Pending` state.
- Problems when deleting VPA: If you delete or disable VPA (Off mode), the resources will remain at the last modified values. This can cause confusion when resource values differ between the Helm charts, the controller, and the pods themselves.
- Using multiple VPA resources for a single pod: This can lead to unpredictable behavior.

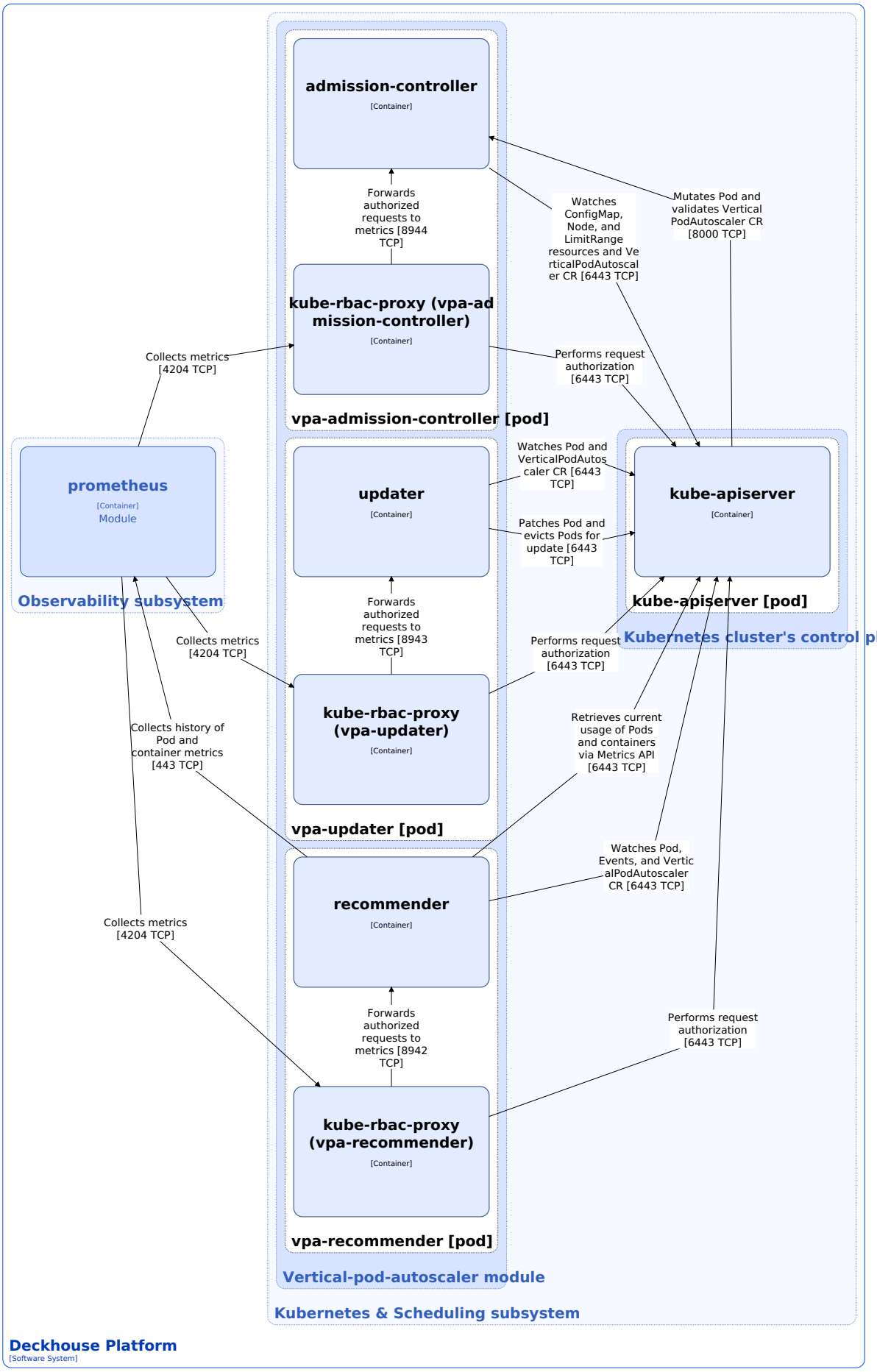
⚠ When using VPA, it is recommended to configure a Pod Disruption Budget.

Module architecture

i The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `vertical-pod-autoscaler` module and its interactions with other DP components are shown in the following diagram:



Module components

The `vertical-pod-autoscaler` module consists of the following components:

1. **Vpa-admission-controller** (Deployment): A VPA controller that handles the VerticalPodAutoscaler

custom resource.

The vpa-admission-controller component performs the following actions:

- Validates VerticalPodAutoscaler custom resources.
- When a Pod is created and the VPA mode is not Off, the controller automatically sets or updates `requests` and `limits` in containers, optimizing them according to the current recommendations. The controller updates `limits` values only if the `spec.resourcePolicy.containerPolicies.controlledValues` parameter in the resource management policy is set to `RequestsAndLimits`.

It consists of the following containers:

- **admission-controller**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to admission-controller metrics. It is an open source project.

2. **Vpa-updater** (Deployment): A VPA component that checks whether pods with VPA have the correct resource settings. Vpa-updater performs in-place resource updates through the `pods/resize` Kubernetes subresource; if that is not possible or does not fit the resource management policy, it evicts the Pod.

   It consists of the following containers:

   - **updater**: Main container.
   - **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to updater metrics. It is an open source project.

3. **Vpa-recommender** (Deployment): A VPA component that calculates recommendations for `requests` based on past and current pod resource consumption.

   Vpa-admission-controller and vpa-updater recalculate `limits` values proportionally to `requests` values if the `spec.resourcePolicy.containerPolicies.controlledValues` parameter in resource management policy is set to `RequestsAndLimits`.

   It consists of the following containers:

   - **recommender**: Main container.
   - **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to recommender metrics. It is an open source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches standard resources such as ConfigMap, Node, LimitRange, and Pod, as well as VerticalPodAutoscaler and VerticalPodAutoscalerCheckpoint custom resources.
   - Retrieves current resource consumption through the Metrics API.
   - Evicts running pods when their resource specifications do not match the recommended values.
   - Authorizes requests for metrics.
2. **Prometheus**: Retrieves the history of pod resource consumption metrics.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates VerticalPodAutoscaler custom resources.
   - Changes `requests` and `limits` in the Pod specification.
2. **Prometheus**: Collects module metrics.

Additional resources

- Enabling vertical scaling
- Module `control-plane-manager` parameters
- Module `vertical-pod-autoscaler` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Adaptive-resource-management module

The `adaptive-resource-management` module lets you automate the selection of resource requests and limits for workloads by leveraging the Vertical Pod Autoscaler (VPA) recommendations.

The module deploys the **AutoVPA** controller based on Goldilocks tailored for Deckhouse Platform (DP). The controller automatically creates and maintains VPA objects for workloads in selected namespaces and provides recommendations on resource configuration.

Main features:

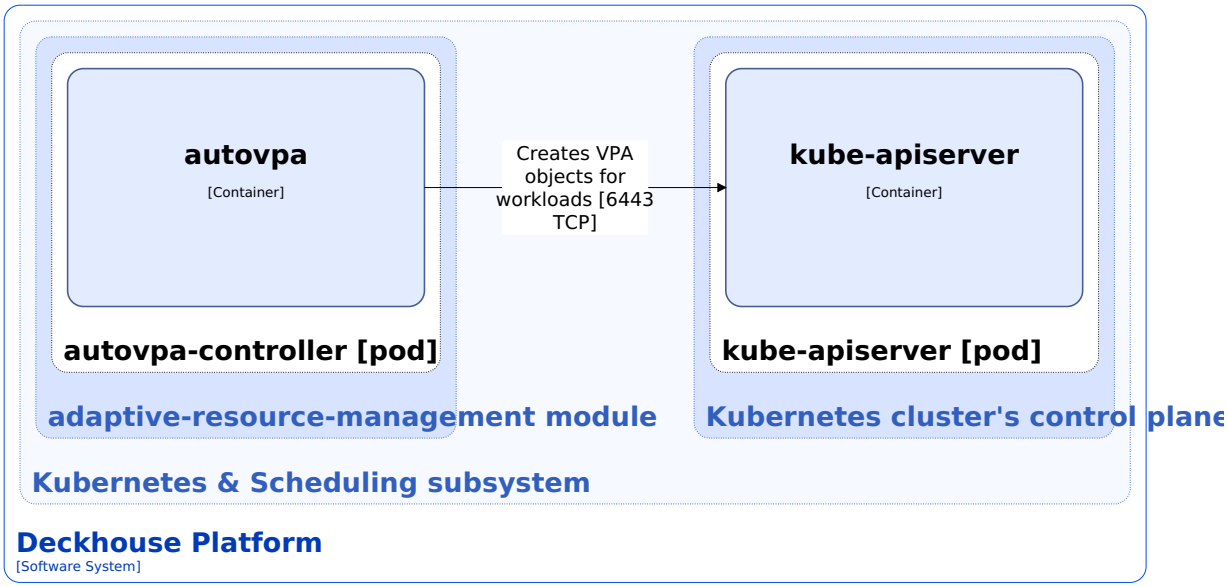
- Automatic creation of VPA objects for Deployments, StatefulSets, DaemonSets, and Jobs in managed namespaces.
- Flexible namespace selection: manage all namespaces, only namespaces with a specific label, or namespaces matching a label selector.
- Combined namespace selection mode using a label selector together with the special label `autovpa.deckhouse.io/enabled`.
- Automatic creation of VPA objects in recommendation-only mode without changing the workload manifests.
- Per-workload and per-namespace VPA tuning via `autovpa.deckhouse.io/*` annotations and labels.
- Minimal resource footprint: the controller runs as a single replica and requires low CPU and memory resources.

For more details about the module configuration and usage examples, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `adaptive-resource-management` module and its interactions with other components of DP are shown in the following diagram:



Module components

The `adaptive-resource-management` module consists of a single **autovpa-controller** component that includes **autovpa**, the main container.

Module interactions

The module interacts with the **kube-apiserver** component for creating VPA objects for workloads.

Additional resources

- Module `adaptive-resource-management` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Descheduler module

The `descheduler` module ensures operation of Descheduler in Deckhouse Platform (DP).

The module analyzes the cluster state periodically and evicts pods that match conditions defined in active strategies. Evicted pods are then scheduled again according to the current cluster state. This helps redistribute workloads in line with the selected strategy.

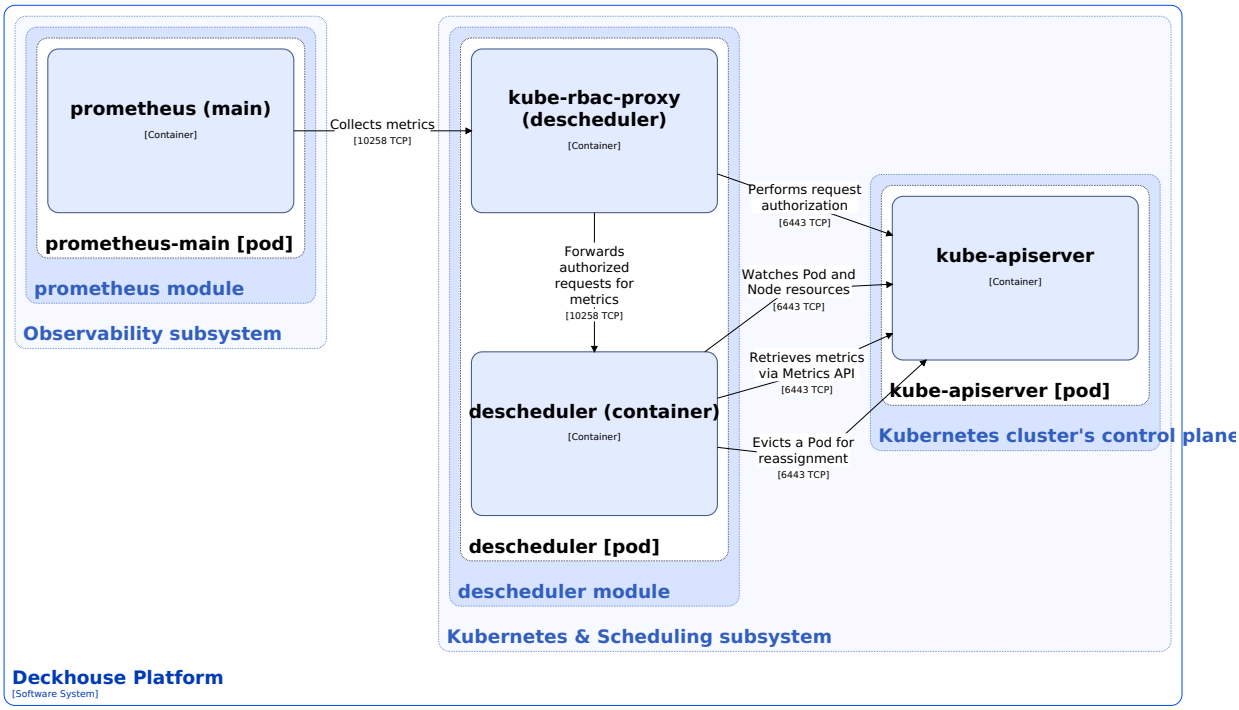
If the Metrics API service is registered in the cluster, the module uses it to obtain information about the current resource consumption.

Strategy parameters are configured using the Descheduler custom resource. A module hook processes these resources and recreates configuration for Descheduler.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `descheduler` module and its interactions with other DP components are shown in the following diagram:



Module components

The `descheduler` module consists of a single **descheduler** component that includes the following containers:

- **descheduler**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides

secure access to descheduler metrics.

Module interactions

The module interacts with the **kube-apiserver** component:

- Watches standard Node and Pod resources.
- Retrieves current resource consumption through the Metrics API.
- Evicts running pods to match active strategies.
- Authorizes requests for metrics.

The **prometheus** component interacts with the module by collecting module metrics.

Additional resources

- Module `descheduler` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Pod-reloader module

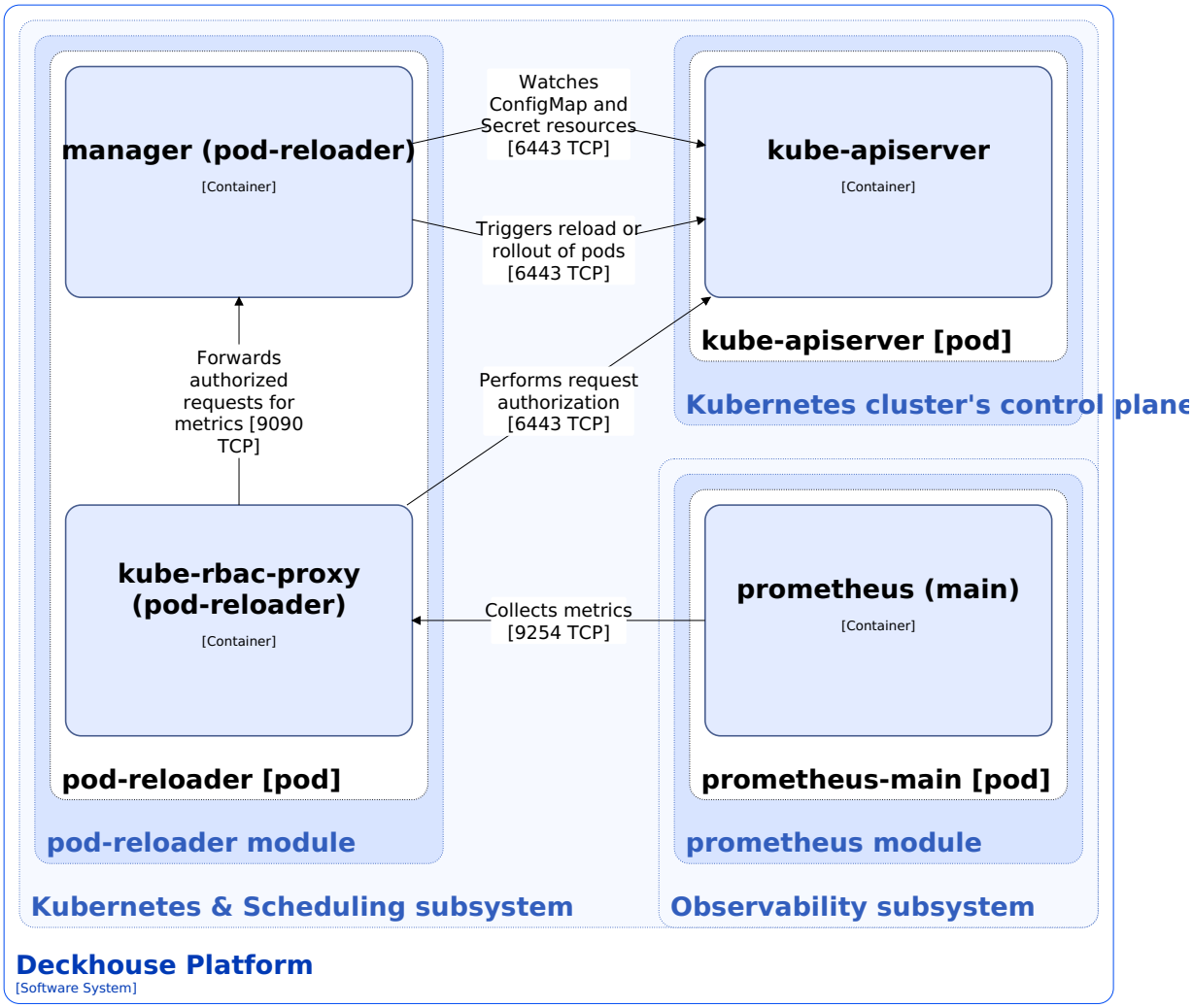
The `pod-reloader` module is based on Reloader. It provides the ability for automatic rollout on ConfigMap or Secret changes. Annotations are used for configuration. The module is running on **system** nodes.

For more details about module configuration and usage examples, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `pod-reloader` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The `pod-reloader` module consists of a single `pod-reloader` component that includes the following containers:

- **manager**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to pod-reloader metrics. It is an open-source project.

Module interactions

The module interacts with the `kube-apiserver` component:

- Watches ConfigMap and Secret resources.
- Triggers reload or rollout of pods.
- Authorizes requests for metrics.

The `prometheus-main` component interacts with the module by collecting module metrics.

Additional resources

- Module `pod-reloader` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster & Infrastructure subsystem

This section describes the architecture of the Cluster & Infrastructure subsystem of the Deckhouse Platform (DP).

The Cluster & Infrastructure subsystem is responsible for the infrastructure layer of Kubernetes cluster management. Cluster node management is implemented using the `node-manager` module, while interaction with IaaS providers is handled by the corresponding `cloud-provider-` family modules.

This section describes:

- The mechanisms for managing all node types used in DP, as well as hybrid node groups and clusters.
- Bashible service, which is a key component of the Cluster & Infrastructure subsystem. Bashible is used by the `node-manager` module to manage node configuration.

The Cluster & Infrastructure subsystem also includes the following modules:

- `chrony`: Provides time synchronization across all cluster nodes.
- `registry-packages-proxy`: Provides an internal proxy server for container registry packages.
- `terraform-manager`: Provides tools for managing Terraform state within a Kubernetes cluster.

- Modules for cloud providers supported by DP:
  - `cloud-provider-aws`
  - `cloud-provider-azure`
  - `cloud-provider-dvp`
  - `cloud-provider-dynamix`
  - `cloud-provider-gcp`
  - `cloud-provider-huaweicloud`
  - `cloud-provider-openstack`
  - `cloud-provider-vcd`
  - `cloud-provider-vsphere`
  - `cloud-provider-yandex`
  - `cloud-provider-zvirt`
- The `infra-provider-baremetal` module, which creates CloudEphemeral nodes on physical servers.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Node management

Node types and addition mechanics

In Deckhouse Platform (DP), nodes are divided into the following types:

- **CloudEphemeral**: Nodes are automatically provisioned, created, and deleted in the configured cloud provider or on physical servers using the `infra-provider-baremetal` module.
- **CloudPermanent**: Persistent nodes created and updated by the `node-manager` module. The nodes differ in that their configuration comes not from the NodeGroup custom resource, but from a special resource <PROVIDER>ClusterConfiguration (for example, AWSClusterConfiguration for AWS). Also, an important difference is that to apply a node configuration, you need to run `dhctl converge` (by running the DP installer). An example of a CloudPermanent node in a cloud cluster is a cluster master node.
- **CloudStatic**: Created manually or by any external tools, located in the same cloud that is integrated with one of the cloud provider modules:
  - CloudStatic nodes have several features related to integration with the cloud provider. These nodes are managed by the `cloud-controller-manager` component, resulting in:
    - Zone and region metadata being automatically added to the Node object.
    - When the virtual machine is deleted in the cloud, the corresponding Node object is also removed from the cluster.
    - CSI driver can be used to attach cloud volumes.
- **Static**: Static node hosted on a bare-metal server or a virtual machine. In the case of a cloud environment, the `cloud-controller-manager` component does not manage such a node — even if one of the cloud providers is enabled. To learn more about working with static nodes, refer to the `node-manager` documentation.

> For Deckhouse Virtualization Platform (DVP), the DVPClusterConfiguration resource is not used. The CloudPermanent node parameters are set in the NodeGroup, DVPInstanceClass, and `cloud-provider-dvp` ModuleConfig resources.

Nodes are added to the cluster by creating a NodeGroup object, which describes the type, parameters, and configuration of the node group. For CloudEphemeral groups, DP interprets this object and automatically creates the corresponding nodes, registering them in the Kubernetes cluster. For other types (e.g., CloudPermanent or Static), node creation and registration must be done manually or via external tools.

Hybrid groups are also supported, where a single NodeGroup can include both Static nodes deployed in the cloud and static (bare-metal or virtual machine) nodes. For example, the main load may be handled by bare-metal servers, while cloud instances are used as scalable additions during peak loads.

Automatic deployment, configuration, and update of Kubernetes nodes

Automatic deployment (in *static/hybrid* — partial), configuration, and software updates work in all clusters regardless of being cloud or bare metal.

Kubernetes node deployment

DP automatically deploys cluster nodes by performing the following **idempotent** operations:

- OS setup and optimization for working with `containerd` and Kubernetes:
  - Required packages are installed from distribution repositories.
  - Kernel parameters, logging settings, log rotation, and other system parameters are configured.
- Installation of required versions of `containerd` and kubelet, and registration of the node in the Kubernetes cluster.
- Kubernetes-api-proxy setup and updating the upstream list for balancing requests from the node to the Kubernetes API.

Maintaining node state

Two types of updates can be applied to maintain the node’s up-to-date state:

- **Regular**: Always applied automatically and do not cause downtime or reboot.
- **Disruptive**: Such as kernel or `containerd` updates, significant kubelet version changes, etc. These can be configured in manual or automatic mode via the `disruptions` section. In automatic mode, a node drain is performed before the update.

Only one node per group is updated at a time, and only if all nodes in the group are available.

DP includes built-in monitoring metrics to track update progress, notify of issues, or prompt for manual approval if needed.

Working with nodes in supported clouds

Each supported cloud provider allows automatic node provisioning. You need to specify required parameters for each node or group.

Depending on the provider, parameters may include:

- Node type or CPU/memory capacity
- Disk size
- Security settings
- Connected networks, etc.

VM creation, startup, and cluster joining are performed automatically.

Cloud node scaling

Two node scaling modes are available:

- **Automatic scaling.**

  When resources are insufficient and pods are in `Pending` state, nodes are added to the group. If there is no load on one or several nodes, they are removed from the cluster. With automatic scaling, a group priority is considered (groups with higher priority are scaled first).

  To enable auto-scaling, specify different non-zero values for `minPerZone` and `maxPerZone` number of nodes per group.

- **Fixed node count.**

  DP will maintain the specified number of nodes (for example, by provisioning new nodes to replace failed ones).

  To disable auto-scaling and maintain a fixed count, use the same value for `minPerZone` and `maxPerZone`.

Working with Static nodes

When working with static nodes, node-manager functions are limited as follows:

- **No node provisioning.** Resource allocation (bare-metal servers, VMs, etc.) is manual. Further configuration (joining the cluster, monitoring, etc.) can be fully or partially automated.
- **No auto-scaling.** Maintaining node count in a group is available using Cluster API Provider Static (via the `staticInstances.count` parameter). DP tries to keep the specified node count in the group, removing excessive nodes and configuring the new ones (from StaticInstance resources in *Pending* state).

Node configuration, draining, adding to the cluster and removal can be performed in one of the following ways:

- **Manually** using prepared scripts.

  To configure a server (VM) and add a node to the cluster, download and run a bootstrap script. It's generated for each static NodeGroup and stored in the secret `d8-cloud-instance-manager/manual-bootstrap-for-<NODEGROUP-NAME>`.

  To remove a node from the cluster and wipe the server (VM), use the `/var/lib/bashible/cleanup_static_node.sh` script (already present on each static node).

- **Automatically** using Cluster API Provider Static.

  CAPS connects to the server (VM) using StaticInstance and SSHCredentials resources, configures and adds the node to the cluster.

  If needed (e.g., if a corresponding StaticInstance resource is removed or the node count is decreased), CAPS connects to the cluster node, drains it, and removes from the cluster.

- **Manually, with transition to automatic management** via CAPS.

  > **i** Available since DP 1.63.

  To transfer an existing cluster node under CAPS control, prepare StaticInstance and SSHCredentials as for automatic management described above, and annotate the StaticInstance resource with `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Node grouping and group management

Grouping and managing nodes as a group means that all group nodes will share metadata from the `NodeGroup` custom resource.

Monitoring features for node groups include:

- Grouped node parameter charts
- Grouped node availability alerts
- Alerts for N unavailable nodes or N% group nodes, etc.

What is the Instance resource

The Instance resource in Kubernetes represents a definition of an ephemeral virtual machine object, without a specific implementation. It's an abstraction used to manage machines created by tools such as `MachineControllerManager` or Cluster API Provider Static.

The object has no specification. Its status includes:

1. Reference to InstanceClass if available.
2. Reference to the Kubernetes Node object.
3. Current machine status.
4. Info on how to check machine creation logs (available during machine creation).

Creating or deleting a machine will create or delete the corresponding Instance resource. You cannot manually create an Instance resource, but you can delete it. This will remove the machine from the cluster (implementation-dependent).

When node reboot is required

Some node configuration changes may require a reboot.

For example, a node reboot is needed after modifying `sysctl` parameters like `kernel.yama.ptrace_scope`.

NodeGroup parameter effects on node updates and restarts

NodeGroup parameter	Disruption update	Node reprovisioning	Kubelet restart
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+
kubernetesVersion	-	-	+
nodeTemplate	-	-	-
static	-	-	+
update.maxConcurrent	-	-	-

For full details, see the NodeGroup custom resource documentation.

When the InstanceClass parameters change, the platform replaces the group machines with new ones. How many instances are replaced at the same time is defined by the `cloudInstances.maxSurgePerZone` and `cloudInstances.maxUnavailablePerZone` parameters. When `instancePrefix` changes in the DP configuration, the platform creates new MachineDeployment objects and removes the old ones.

When an update requires a node disruption, a pod eviction process is initiated. If a pod cannot be evicted, the eviction attempt is retried every 20 seconds for up to 5 minutes global timeout. After that, pods that have not been evicted are forcefully removed.

Additional resources

- Module `cloud-provider-aws` provider configuration
- Module `cloud-provider-dvp` custom resources
- Module `cloud-provider-dvp` parameters
- Module `infra-provider-baremetal` documentation
- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Bashible

Bashible scripts and the bashible service

Node management functions are implemented using specially prepared bash scripts called **bashible**. The service that runs on cluster nodes and executes these scripts is also called bashible. A collection of scripts is referred to as a bundle.

4 bundles are used:

- Bashible installation scripts
- Bootstrap scripts for the first node
- Node configuration scripts for a specific cloud provider (for example, AWS)
- Common scripts

The scripts are implemented as *gotemplate* templates, which allows flexible node configuration depending on the node group. The scripts must be written in such a manner so that they can be safely re-executed in case of failure or repeated runs. An individual script is referred to as a step.

Main node configuration stages:

- Configuring the NodeUser to ensure access to the node.
- Installing CA certificates.
- Creating the `/opt/deckhouse/bin` directory, adding it to `PATH`, and placing required binaries there.
- Downloading required packages from `registrypackages`.
- Installing and configuring the containerd CRI.
- Downloading and configuring **kubernetes-api-proxy**. This component provides access to the Kubernetes API and is implemented as an NGINX instance with upstream servers pointing to master nodes. This ensures high availability of the API in case one master node is unavailable, as well as load balancing.
- Installing, configuring, and starting kubelet.
- Starting the bashible service, which runs `bashible.sh` every minute.
- Rebooting the node if required.

Bashible-api-server

Due to the large number of bashible script variations for different supported operating systems, storing all versions in etcd is not feasible because of key size limitations and the additional load on etcd. For this reason, the **bashible-api-server** component was developed to generate bashible scripts from templates stored in Custom Resources.

Bashible-api-server is a Kubernetes Extension API Server deployed on master nodes.

When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. The interactions between bashible and bashible-api-server are shown in the architecture diagrams of the `node-manager` module (for example, in the CloudEphemeral nodes diagram).

Bashible-api-server returns the following resources:

- The **second-phase bootstrap script** downloaded during the first phase.
- **Bashibles**: The `bashible.sh` script.
- **Nodegroupbundles**: Rendered bundle containing the set of scripts for bootstrapping and configuring a node.

These resources can be retrieved either via the Kubernetes API or using `kubectl` by specifying the node group name:

- `kubectl get bootstrap.bashible.deckhouse.io master -o yaml`
- `kubectl get bashibles.bashible.deckhouse.io master -o yaml`
- `kubectl get nodegroupbundles.bashible.deckhouse.io master -o yaml`

Bashible-api-server calculates a checksum of all scripts associated with a node group. This is required to implement the update mechanism and ensure correct node group status updates. The checksums are stored in the `d8-cloud-instance-manager/configuration-checksums` secret. A checksum change triggers the re-execution of bashible scripts on nodes when the configuration is modified. In addition, the checksum of the bashible service is reset every 4 hours to force periodic re-execution of bashible.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Node-manager module

Cluster nodes are managed by the `node-manager` module.

For a detailed description of the module’s functions, configuration options, and usage examples, refer to the corresponding documentation section.

Module architecture

The module architecture varies depending on the node type and differs in terms of its component composition. The following pages describe the module architecture for different node types:

- Managing CloudEphemeral nodes
- Managing CloudPermanent nodes
- Managing CloudStatic nodes
- Managing Static nodes
- Managing hybrid node groups and clusters

Additional resources

- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

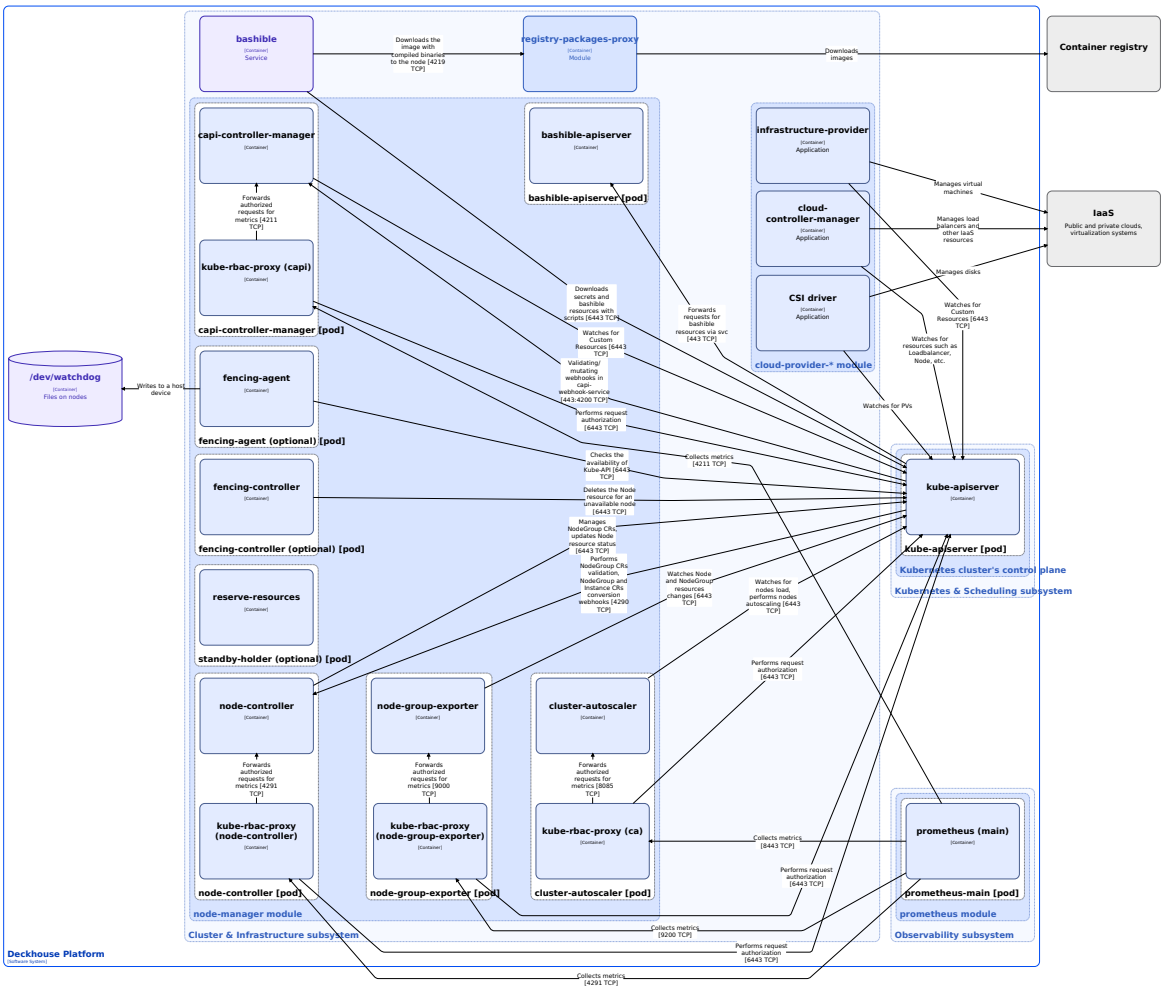
CloudEphemeral node management

This page describes the architecture of the `node-manager` module for CloudEphemeral nodes.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Platform (DP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudEphemeral nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.
2. **Node-controller** (Deployment): A controller that manages NodeGroup custom resources lifecycle. Node-controller performs the following operations:
   - Manages NodeGroup custom resources lifecycle.
   - Implements NodeGroup custom resources validating webhooks using the Validating Admission Controllers mechanism.
   - Implements NodeGroup and Instance custom resources conversion webhooks.
   - Cleans up Node resource labels and taints that remain after the bashible first run to initialize the node.
   - Ensures draining a node.
   - Applies labels, taints and annotations from the `spec.nodeTemplate` section of a NodeGroup custom resource to all Node resources belonging to it.
   - Calculates and updates a NodeGroup custom resource `status` subresource based on aggregated information obtained from the corresponding Node resources and infrastructure custom resources.
   - Sets the `spec.providerId = "static://"` attribute for Static type Node resources if it is missing.
   - Manages the node update lifecycle: approving updates, handling node disruptions, draining a node and cleanup after a successful update.

   The component includes:
   - **node-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.
3. **Node-group-exporter** (Deployment): A component that exports NodeGroup resource metrics in Prometheus format, containing information about the number of nodes in each node group: the total number, the number of nodes in the `Ready` status, the number of nodes in error, the minimum and maximum number of nodes in the group, etc.

   The component includes:
   - **node-group-exporter**: Main container.
   - **kube-rbac-proxy** Sidecar container providing an RBAC-based authorization proxy for secure access to exporter metrics.
4. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster.

   The component includes:
   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.
5. **Cluster-autoscaler** (Deployment): An additional Kubernetes component that automatically adjusts the number of nodes in the cluster based on workload. For more details, refer to the node management documentation section.

   The component includes:
   - **cluster-autoscaler**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cluster-autoscaler metrics.
6. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.
7. **Standby-holder** (Deployment): A pod used to reserve nodes. When the `spec.cloudinstances.standby` parameter is enabled in the NodeGroup custom resource, standby nodes are created in all configured zones.

A standby node is a cluster node with pre-reserved resources available for immediate scaling. This allows cluster-autoscaler to schedule workloads without waiting for node initialization, which may take several minutes.

The standby-holder pod does not perform useful work. It simply reserves resources to prevent cluster-autoscaler from deleting temporarily unused nodes.

The pod has the lowest PriorityClass and is evicted when real workloads are scheduled. For details on pod priority and preemption, refer to the Kubernetes documentation.

The component includes a single container **reserve-resources**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Retrieves the `kube-system/d8-node-manager-cloud-provider` Secret for cloud connectivity.
   - Works with Cluster API custom resources.
   - Manages Node and NodeGroup resources.
   - Monitors node load.
   - Performs node autoscaling.
   - Authorizes metric requests.
2. Node filesystem:
   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

> **i**
> The module interacts with the `cloud-provider` module via kube-apiserver using the `kube-system/d8-node-manager-cloud-provider` Secret to obtain cloud connection settings and create CloudEphemeral nodes. The `cloud-provider` module also provides provider-specific Cluster API custom resource templates to `node-manager`.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Executes validating and conversion webhooks of node-controller.
   - Executes mutating and validating webhooks of capi-controller-manager.
   - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
   - Collects metrics from `node-manager` module components.

Architecture features specific to CloudEphemeral nodes

1. Nodes are ephemeral and automatically created and deleted by the module.
2. A configured cloud provider module (`cloud-provider-*`) is required for interaction with cloud infrastructure. It also includes csi-driver and cloud-controller-manager. Nodes on physical servers are created using the `infra-provider-baremetal` module, which does not include csi-driver or cloud-controller-manager.
3. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the `cloud-provider` module.
4. **Cluster-autoscaler** enables node autoscaling.
5. Node reservation is supported.

Additional resources

- Module `infra-provider-baremetal` documentation
- Module `node-manager` documentation

> **i**
> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

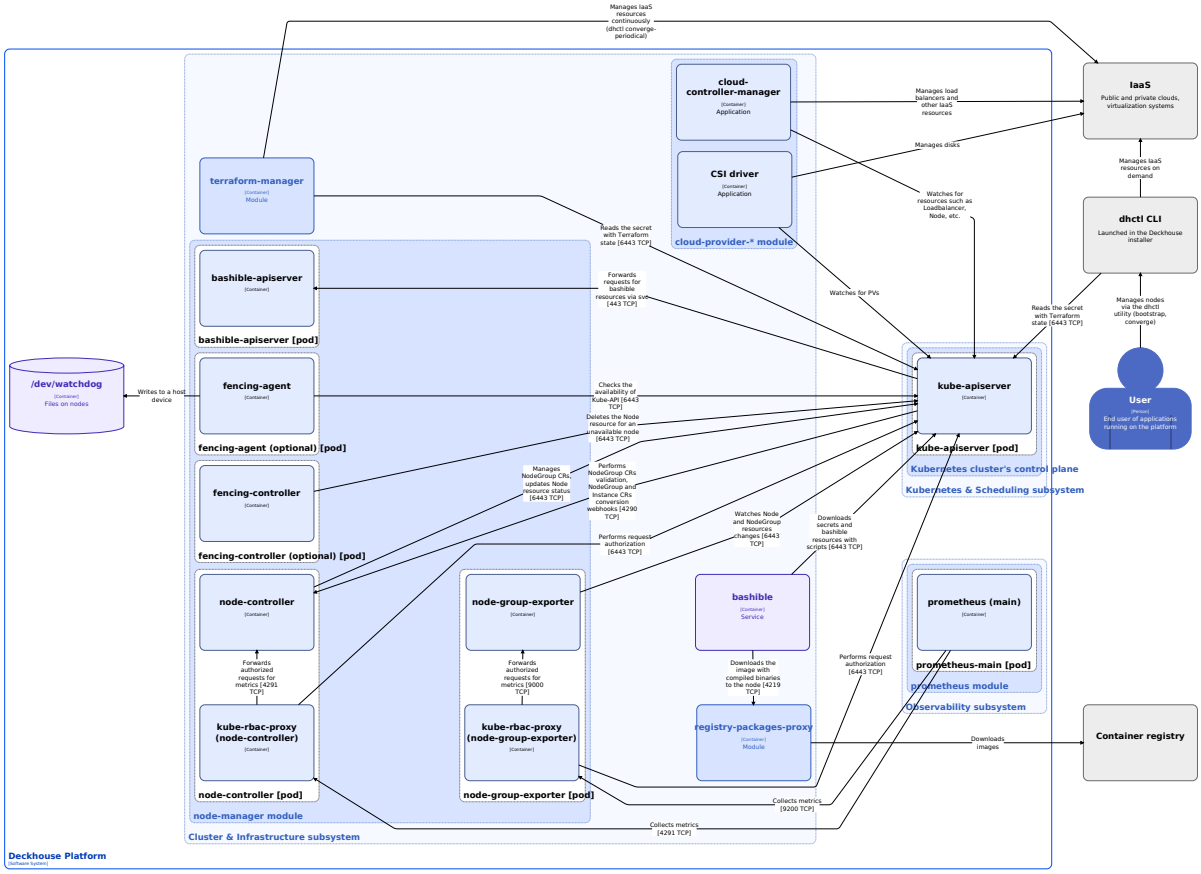
CloudPermanent node management

This page describes the architecture of the `node-manager` module for CloudPermanent nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Platform (DP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudPermanent nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.
2. **Node-controller** (Deployment): A controller that manages NodeGroup custom resources lifecycle. Node-controller performs the following operations:
   - Manages NodeGroup custom resources lifecycle.
   - Implements NodeGroup custom resources validating webhooks using the Validating Admission Controllers mechanism.
   - Implements NodeGroup and Instance custom resources conversion webhooks.
   - Cleans up Node resource labels and taints that remain after the bashible first run to initialize the node.

- Ensures draining a node.
- Applies labels, taints and annotations from the `spec.nodeTemplate` section of a NodeGroup custom resource to all Node resources belonging to it.
- Calculates and updates a NodeGroup custom resource `status` subresource based on aggregated information obtained from the corresponding Node resources and infrastructure custom resources.
- Sets the `spec.providerId = "static://"` attribute for Static type Node resources if it is missing.
- Manages the node update lifecycle: approving updates, handling node disruptions, draining a node and cleanup after a successful update.

The component includes:

- **node-controller**: Main container.
- **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

3. **Node-group-exporter** (Deployment): A component that exports NodeGroup resource metrics in Prometheus format, containing information about the number of nodes in each node group: the total number, the number of nodes in the `Ready` status, the number of nodes in error, the minimum and maximum number of nodes in the group, etc.

   The component includes:

   - **node-group-exporter**: Main container.
   - **kube-rbac-proxy** Sidecar container providing an RBAC-based authorization proxy for secure access to exporter metrics.

4. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages Node and NodeGroup resources.
   - Authorizes metric requests.
2. Node filesystem:
   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Executes validating and conversion webhooks of node-controller.
   - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
   - Collects metrics from `node-manager` module components.

Architecture features specific to CloudPermanent nodes

1. Nodes are persistent and are created, managed, and deleted by the user. Node management is performed not directly in the infrastructure but via the **dhctl** utility executed as part of the DP installer.
2. `Terraform-manager` is a module used for automated management of cloud infrastructure resources. It checks the Terraform state and applies non-destructive changes to infrastructure resources. The module architecture is described on the corresponding documentation page.
3. **Csi-driver** is used to provision disks in the cloud infrastructure.
4. **Cloud-controller-manager** is used to provision load balancers and other infrastructure resources according to its specification.
5. **Infrastructure-provider** is not required. All node management operations are performed by the user via the **dhctl** utility and the `terraform-manager` module.
6. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

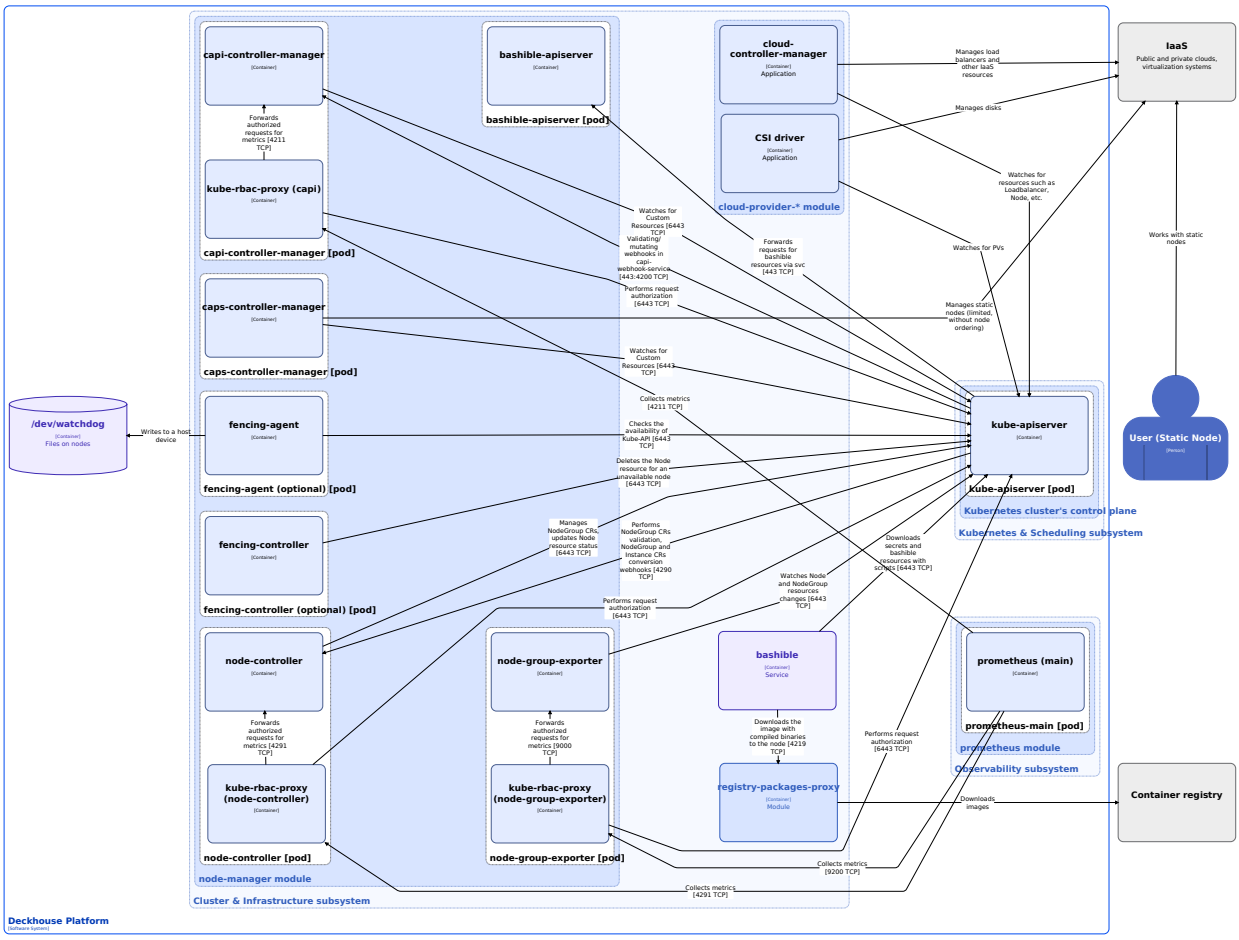
CloudStatic node management

This page describes the architecture of the `node-manager` module for CloudStatic nodes.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Platform (DP) components are shown in the following diagram:



Module components

> Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudStatic nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates

bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.

2. **Node-controller** (Deployment): A controller that manages NodeGroup custom resources lifecycle. Node-controller performs the following operations:

   - Manages NodeGroup custom resources lifecycle.
   - Implements NodeGroup custom resources validating webhooks using the Validating Admission Controllers mechanism.
   - Implements NodeGroup and Instance custom resources conversion webhooks.
   - Cleans up Node resource labels and taints that remain after the bashible first run to initialize the node.
   - Ensures draining a node.
   - Applies labels, taints and annotations from the `spec.nodeTemplate` section of a NodeGroup custom resource to all Node resources belonging to it.
   - Calculates and updates a NodeGroup custom resource `status` subresource based on aggregated information obtained from the corresponding Node resources and infrastructure custom resources.
   - Sets the `spec.providerId = "static://"` attribute for Static type Node resources if it is missing.
   - Manages the node update lifecycle: approving updates, handling node disruptions, draining a node and cleanup after a successful update.

   The component includes:

   - **node-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

3. **Node-group-exporter** (Deployment): A component that exports NodeGroup resource metrics in Prometheus format, containing information about the number of nodes in each node group: the total number, the number of nodes in the `Ready` status, the number of nodes in error, the minimum and maximum number of nodes in the group, etc.

   The component includes:

   - **node-group-exporter**: Main container.
   - **kube-rbac-proxy** Sidecar container providing an RBAC-based authorization proxy for secure access to exporter metrics.

4. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster.

   The component includes:

   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

5. **Caps-controller-manager** (Deployment): CAPI Provider Static (CAPS), an implementation of a provider for declarative management of static nodes (bare-metal servers or virtual machines) in the Kubernetes Cluster API project. It operates as an extension to capi-controller-manager.

   CAPS provides an additional abstraction layer over the existing DP mechanism for automatic configuration and cleanup of static nodes using scripts generated for each node group. The component is not tied to a specific cloud provider. For more details, refer to the `node-manager` documentation.

6. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Works with Cluster API custom resources.
   - Manages Node and NodeGroup resources.
   - Authorizes metric requests.

2. Node filesystem:

   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

3. Infrastructure:
    - Manages static nodes (partially, without provisioning).

The following external components interact with the module:

1. **Kube-apiserver**:
    - Executes validating and conversion webhooks of node-controller.
    - Executes mutating and validating webhooks of capi-controller-manager.
    - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
    - Collects metrics from `node-manager` module components.

Architecture features specific to CloudStatic nodes

1. Users create and configure nodes in the following ways:
    - Manually, using bashible scripts preconfigured in DP.
    - Manually, with the following node handover to CAPS for automated management.
    - Automatically, using CAPS.
2. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the infrastructure provider (CAPS).
3. **Caps-controller-manager**: Component managing static nodes (partially, without provisioning).
4. **Csi-driver** is used to provision disks in the cloud infrastructure.
5. **Cloud-controller-manager** is used to provision load balancers and other infrastructure resources according to its specification.
6. **Infrastructure-provider** of a specific cloud is not required, as CAPS acts on its behalf.
7. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

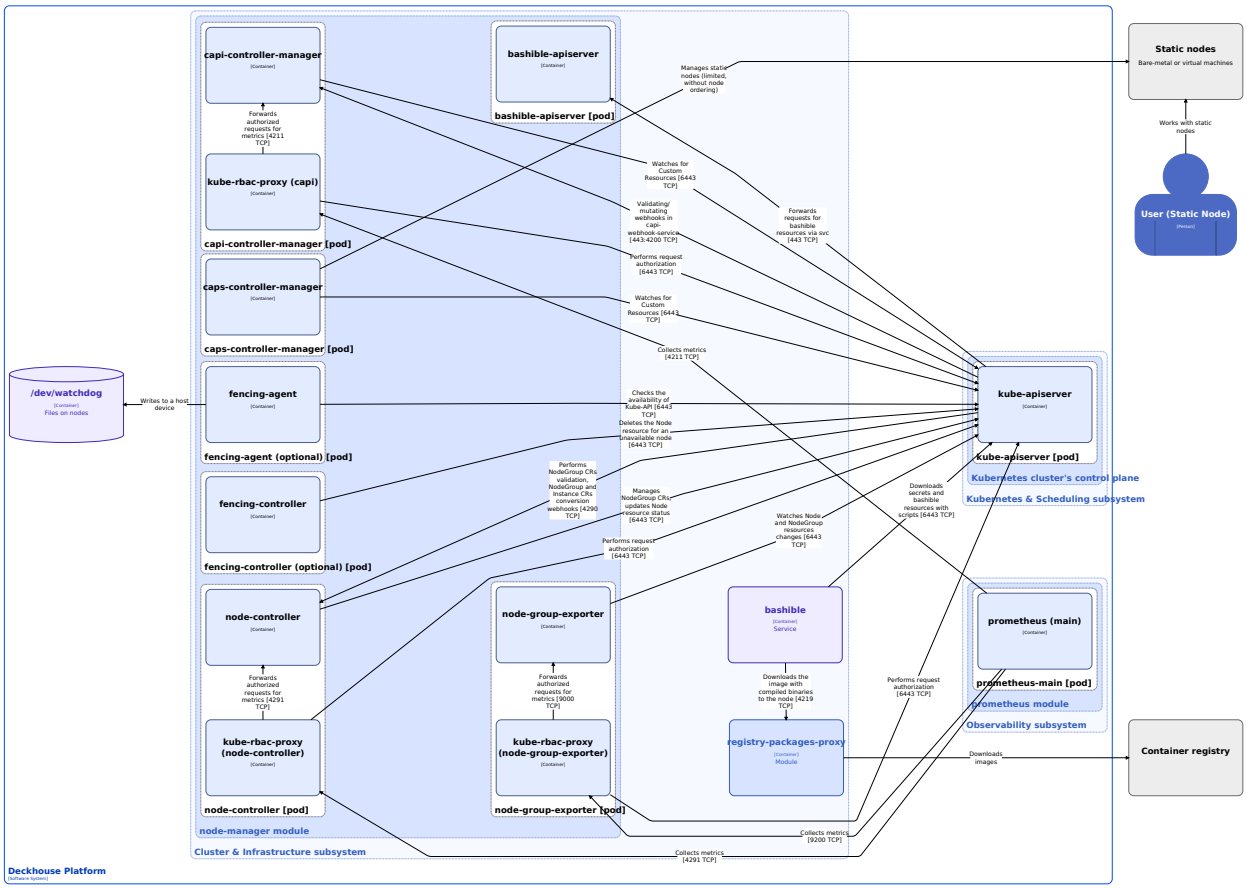
Static node management

This page describes the architecture of the `node-manager` module for Static nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Platform (DP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing Static nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.

2. **Node-controller** (Deployment): A controller that manages NodeGroup custom resources lifecycle. Node-controller performs the following operations:

   - Manages NodeGroup custom resources lifecycle.
   - Implements NodeGroup custom resources validating webhooks using the Validating Admission Controllers mechanism.
   - Implements NodeGroup and Instance custom resources conversion webhooks.
   - Cleans up Node resource labels and taints that remain after the bashible first run to initialize the node.
   - Ensures draining a node.
   - Applies labels, taints and annotations from the `spec.nodeTemplate` section of a NodeGroup custom resource to all Node resources belonging to it.
   - Calculates and updates a NodeGroup custom resource `status` subresource based on aggregated information obtained from the corresponding Node resources and infrastructure custom resources.
   - Sets the `spec.providerId = "static://"` attribute for Static type Node resources if it is missing.
   - Manages the node update lifecycle: approving updates, handling node disruptions, draining a node and cleanup after a successful update.

   The component includes:

   - **node-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

3. **Node-group-exporter** (Deployment): A component that exports NodeGroup resource metrics in

Prometheus format, containing information about the number of nodes in each node group: the total number, the number of nodes in the `Ready` status, the number of nodes in error, the minimum and maximum number of nodes in the group, etc.

The component includes:

- **node-group-exporter**: Main container.
- **kube-rbac-proxy** Sidecar container providing an RBAC-based authorization proxy for secure access to exporter metrics.

4. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster.

   The component includes:

   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

5. **Caps-controller-manager** (Deployment): CAPI Provider Static (CAPS), an implementation of a provider for declarative management of static nodes (bare-metal servers or virtual machines) in the Kubernetes Cluster API project. It operates as an extension to capi-controller-manager.

   CAPS provides an additional abstraction layer over the existing DP mechanism for automatic configuration and cleanup of static nodes using scripts generated for each node group. The component is not tied to a specific cloud provider. For more details, refer to the `node-manager` documentation.

6. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Works with Cluster API custom resources.
   - Manages Node and NodeGroup resources.
   - Authorizes metric requests.
2. Node filesystem:
   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Executes validating and conversion webhooks of node-controller.
   - Executes mutating and validating webhooks of capi-controller-manager.
   - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
   - Collects metrics from `node-manager` module components.

Architecture features specific to Static nodes

1. Users create and configure nodes in the following ways:
   - Manually, using bashible scripts preconfigured in DP.
   - Manually, with the following node handover to CAPS for automated management.
   - Automatically, using CAPS.
2. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the infrastructure provider, which is deployed by the specific cloud provider module (CAPS for static nodes).
3. **Caps-controller-manager**: Component managing static nodes (partially, without provisioning).
4. Static nodes can be used not only on bare metal, but in cloud as well. In that case, such a node is not managed by cloud-controller-manager, even if one of the cloud providers is enabled. Csi-driver is not installed on such nodes.
5. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Hybrid node group and cluster management

It is important to distinguish between hybrid node groups and hybrid clusters:

- **Hybrid node groups** may include both Static nodes deployed in the cloud and servers deployed in the customer's on-premises data center (bare metal or virtual machines).

  For example, the primary workload may run on bare-metal servers, while cloud instances are used as a scalable addition during peak loads. Since nodes of the same type (Static) are used both in the customer's data center and in the cloud, the architecture of the `node-manager` module in this case corresponds to the Static node variant. The only mandatory requirement is the presence of L3 connectivity between the on-premises data center and the cloud. For example, Yandex Cloud provides a Cloud Interconnect mechanism.

- **Hybrid clusters** are DP clusters that simultaneously include:

  - NodeGroup with Static nodes deployed in the customer's on-premises data center (bare metal or virtual machines).
  - NodeGroup with CloudEphemeral nodes deployed in the cloud.

  In this case, the components required to manage both node types are deployed. The architecture of the `node-manager` module corresponds to the Static node variant. In addition, components required for CloudEphemeral nodes are deployed:

  - **cloud-provider**: Provides interaction with the cloud infrastructure. A configured provider for the corresponding cloud is required. It also includes csi-driver and cloud-controller-manager.
  - **cluster-autoscaler**: Provides automatic scaling of cluster nodes.

  As with hybrid node groups, hybrid clusters require L3 connectivity between the on-premises data center and the cloud.

Additional resources

- Module `node-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-aws module

The `cloud-provider-aws` module is responsible for interacting with the Amazon Web Services cloud resources. It allows the `node-manager` module to use AWS resources for provisioning nodes for the specified node group.

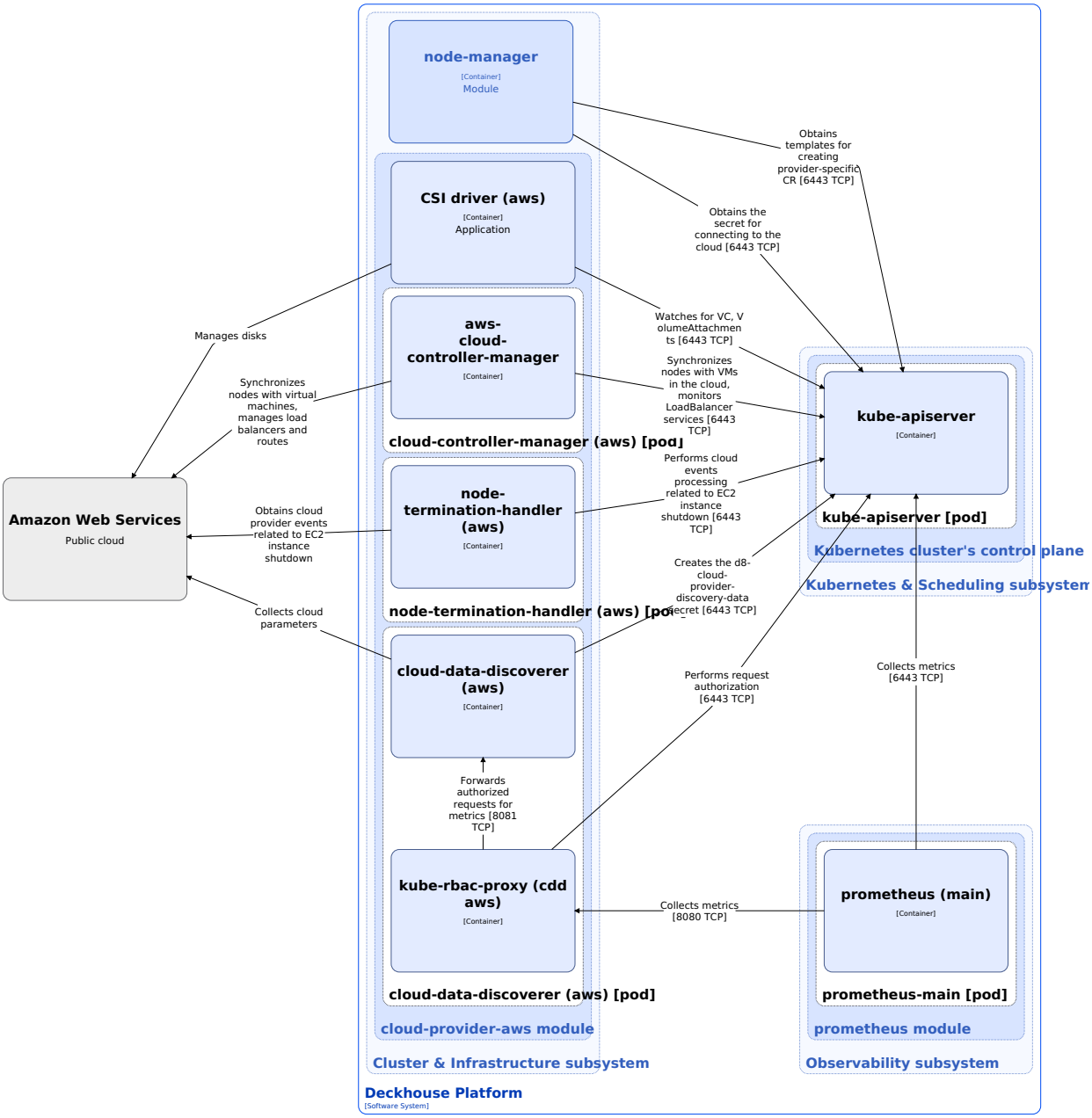
For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-aws` module and its interactions with other components

of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for AWS. It provides interaction with the AWS cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **aws-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-aws` module.

   It consists of the following containers:

- **cloud-data-discoverer**: Main container.
- **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (aws)**: It is an implementation of the CSI driver for AWS. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.

4. **Node-termination-handler**: AWS Node Termination Handler. It is responsible for gracefully handling the termination of EC2 instances in the Kubernetes control plane.

   Node-termination-handler processes the following AWS events:

   - EC2 maintenance events
   - EC2 Spot interruptions
   - EC2 Instance Rebalance Recommendation
   - ASG Scale-In
   - ASG AZ Rebalance
   - EC2 Instance Termination

   It consists of a single container:

   - **node-termination-handler**.

   After receiving an event when it is necessary to drain and then to delete a node, for example, in case of a regular interruption of a spot instance, node-termination-handler sets the `aws-node-termination-handler/spot-itn` taint for the corresponding node.

   When an EC2 Instance Rebalance Recommendation arrives, the `node-termination-handler` component applies the `aws-node-termination-handler/rebalance-recommendation` taint with `effect: NoSchedule` to the CloudEphemeral node. This taint prevents the scheduler from placing new workloads on the node unless they have a matching toleration.

   When either taint is detected, a node drain itself, and then node removal are performed by the `node-manager` module using the module hooks mechanics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.

2. **Amazon Web Services**:

   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Collects cloud events of EC2 instance termination.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-aws` module provides `node-manager` with following artifacts:

   - Provider-specific custom resource templates to be used by `cloud-provider-aws` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.

2. The `cloud-provider-aws` module provides Terraform/OpenTofu components for AWS cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:

   - Terraform/OpenTofu provider.
   - Terraform modules.

   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other.

Each layout should have three modules:

- `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
- `master-node`
- `static-node`.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-azure module

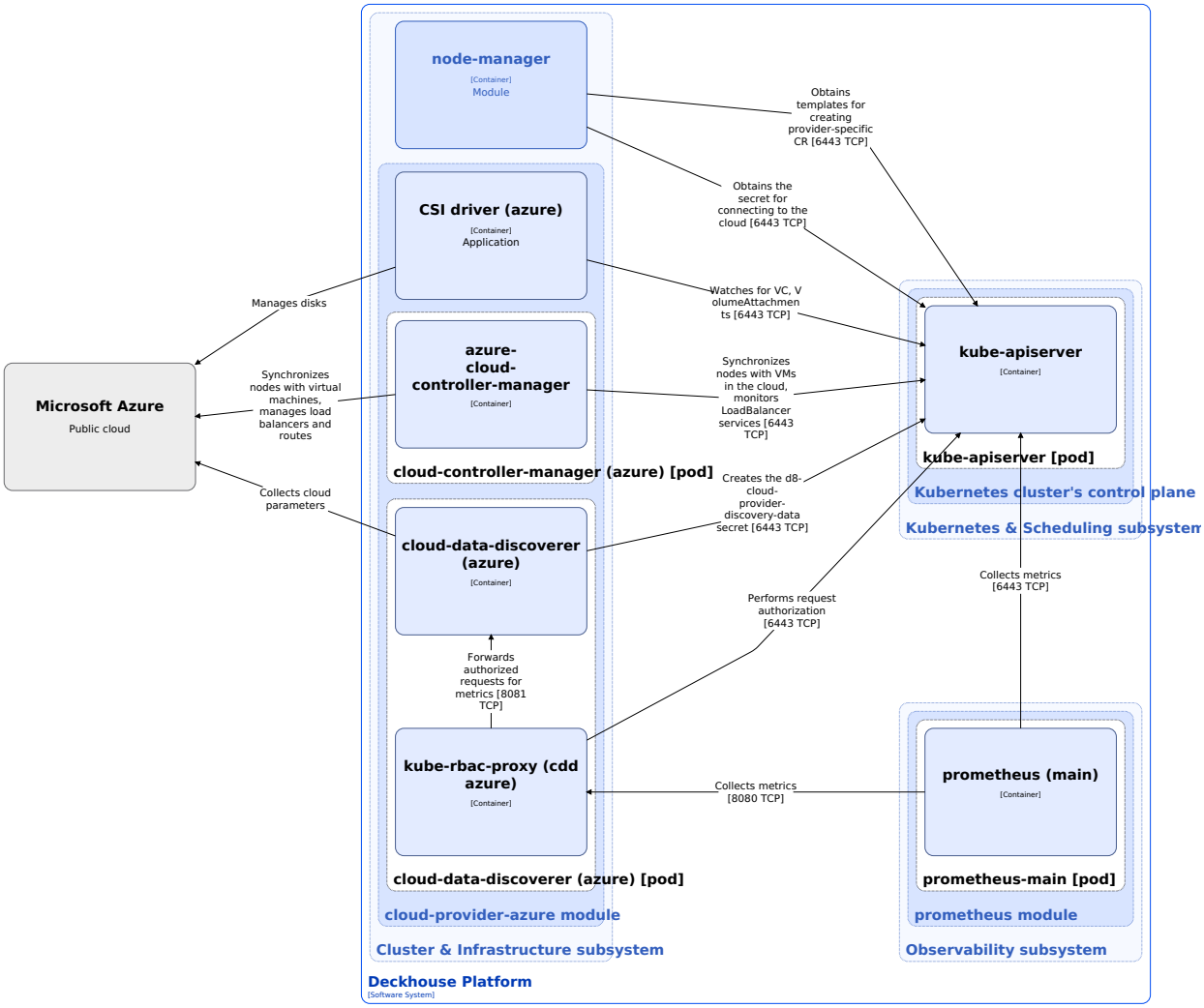
The `cloud-provider-azure` module is responsible for interacting with the Microsoft Azure cloud resources. It allows the `node-manager` module to use Microsoft Azure resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-azure` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for Microsoft Azure. It provides interaction with the Azure cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **azure-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-azure` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (azure)**: It is an implementation of the CSI driver for Microsoft Azure. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Microsoft Azure**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-azure` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-azure` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-azure` module provides Terraform/OpenTofu components for Microsoft Azure cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-azure` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-dvp module

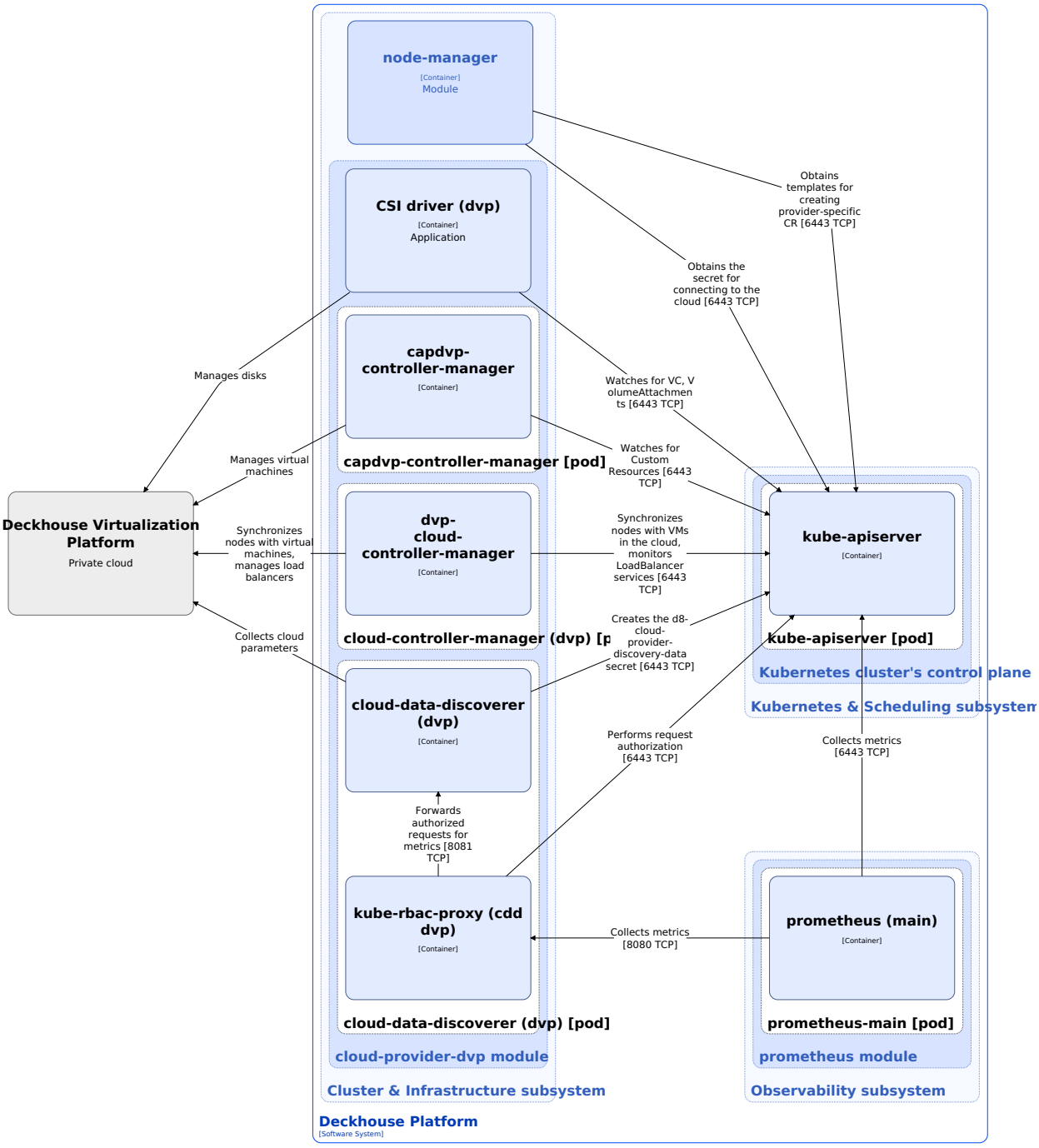
The `cloud-provider-dvp` module is responsible for interacting with the Deckhouse Virtualization Platform (DVP) cloud resources. It allows the `node-manager` module to use DVP resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-dvp` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capdvp-controller-manager**: Kubernetes Cluster API Provider for DVP. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, DVP in this case. Capdvp-controller-manager works with the following custom resources:

- DeckhouseCluster: Description of a DVP-based cluster.
- DeckhouseMachineTemplate: Template describing the characteristics of the machines created in the cloud.
- DeckhouseMachine: Description of the characteristics of a machine created based on DeckhouseMachineTemplate.

It consists of a single container:

- **capdvp-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for DVP. It provides interaction with the DVP cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **dvp-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-dvp` module. For example, for DVP, these are parameters such as a list of availability zones, StorageClass resources, etc.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (DVP)**: It is an implementation of the CSI driver for DVP. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (DVP) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles DeckhouseCluster, DeckhouseMachineTemplate, and DeckhouseMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **DVP cloud**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-dvp` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-dvp` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are

registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.

2. The `cloud-provider-dvp` module provides Terraform/OpenTofu components for DVP cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
    - Terraform/OpenTofu provider.
    - Terraform modules.
    - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
        - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
        - `master-node`
        - `static-node`

Additional resources

- Module `cloud-provider-dvp` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-dynamix module

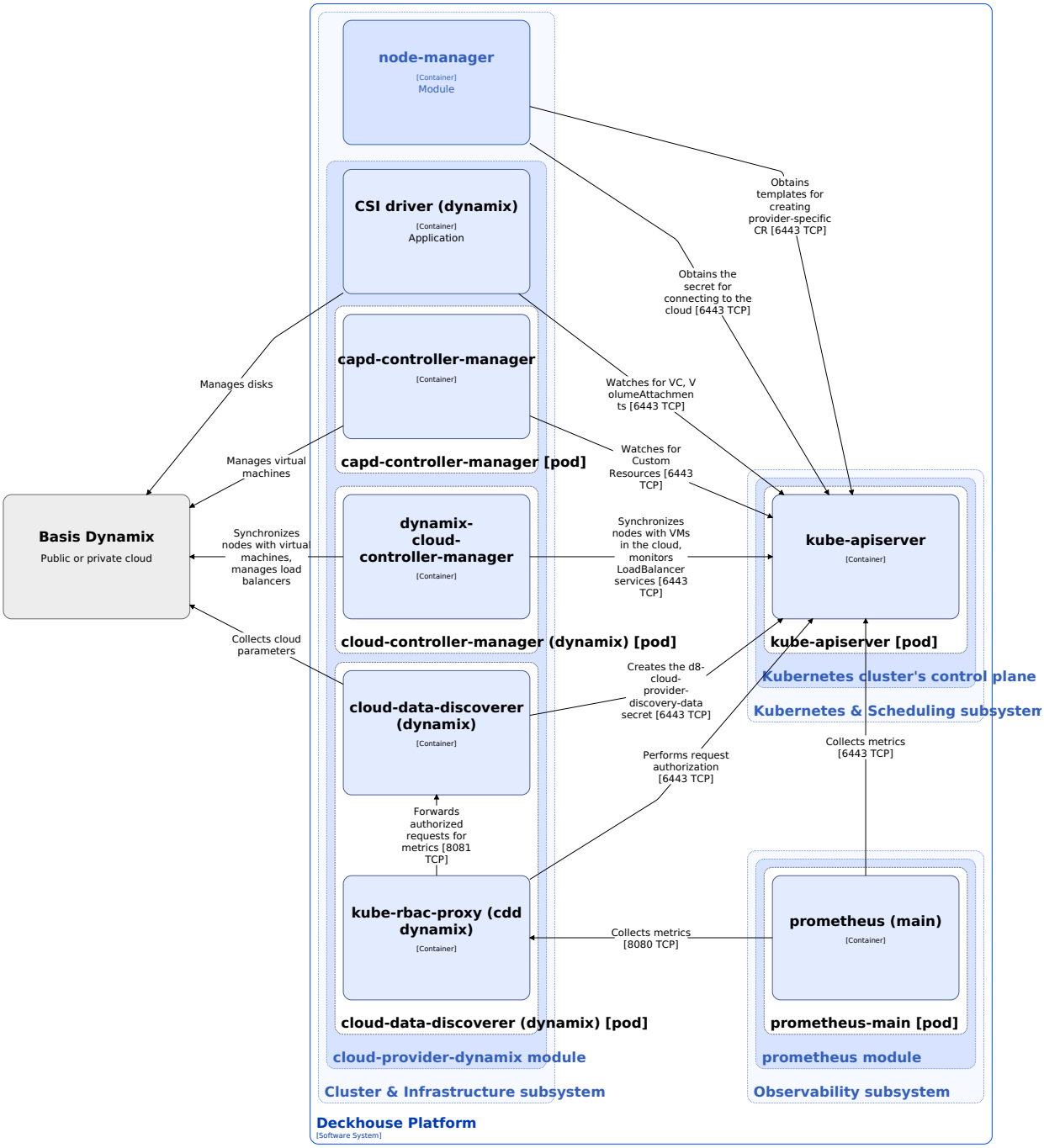
The `cloud-provider-dynamix` module is responsible for interacting with the Basis Dynamix cloud resources. It allows the `node-manager` module to use Basis Dynamix resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-dynamix` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capd-controller-manager**: Kubernetes Cluster API Provider for Basis Dynamix. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, Basis Dynamix in this case. Capd-controller-manager works with the following custom resources:

   - DynamixCluster: Description of a Basis Dynamix-based cluster.
   - DynamixMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - DynamixMachine: Description of the characteristics of a machine created based on DynamixMachineTemplate.

   It consists of a single container:

   - **capd-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Basis Dynamix. It provides interaction with the Basis Dynamix cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud

provider. To do this:

- It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
- It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.

- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **dynamix-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-dynamix` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (dynamix)**: It is an implementation of the CSI driver for Basis Dynamix. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (dynamix) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles DynamixCluster, DynamixMachineTemplate, and DynamixMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.

2. **Basis Dynamix cloud**:

   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-dynamix` module provides `node-manager` with following artifacts:

   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-dynamix` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.

2. The `cloud-provider-dynamix` module provides Terraform/OpenTofu components for Basis Dynamix cloud used when building the dhctl executable file for the `terraform-manager` module, such as:

   - Terraform/OpenTofu provider.
   - Terraform modules.

   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:

     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`

    - `static-node`

Additional resources

- Module `cloud-provider-dynamix` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-gcp module

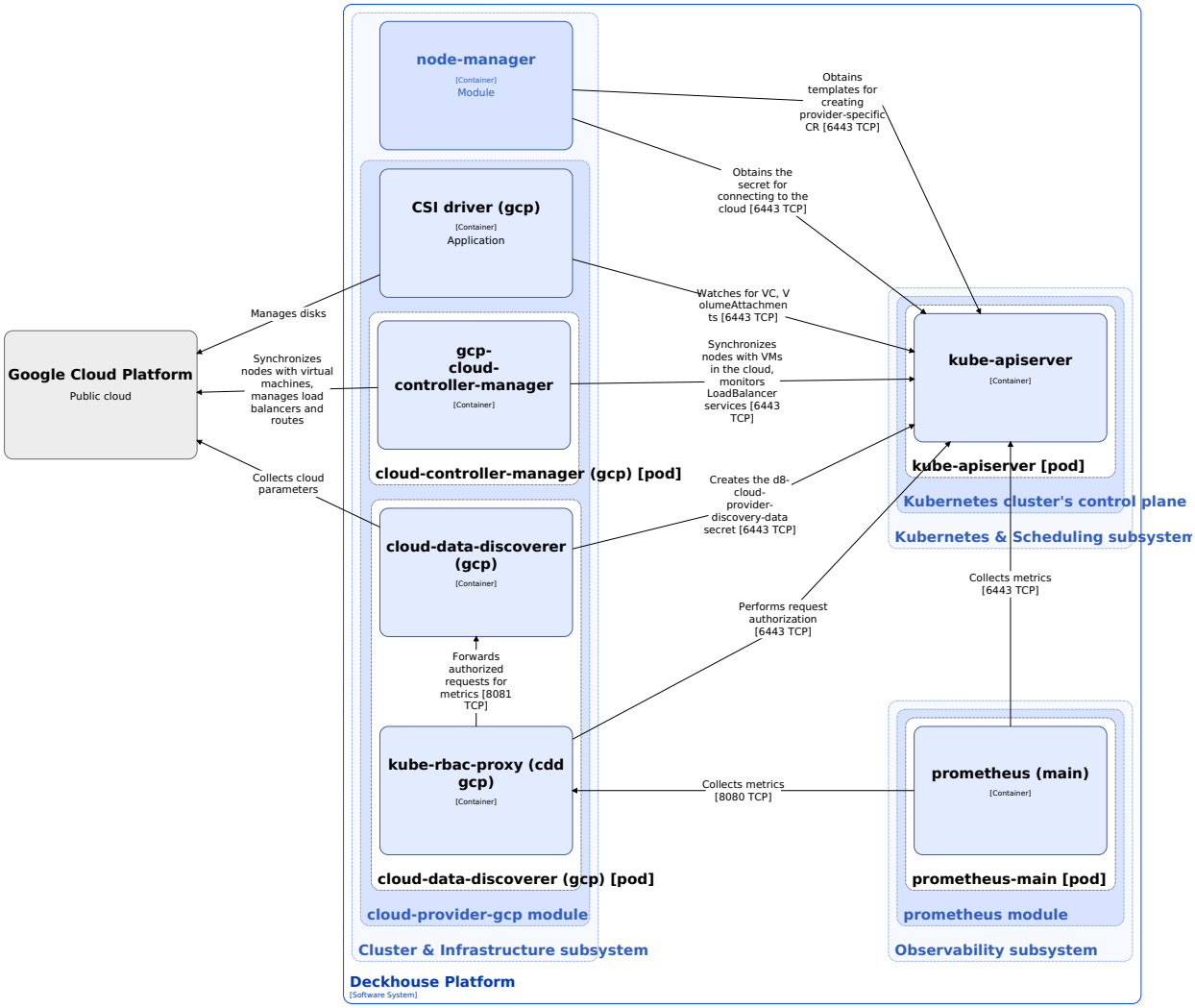
The `cloud-provider-gcp` module is responsible for interacting with the Google Cloud Platform cloud resources. It allows the `node-manager` module to use GCP resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-gcp` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for GCP. It provides interaction with the GCP cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **gcp-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-gcp` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (gcp)**: It is an implementation of the CSI driver for GCP. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Google Cloud Platform**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-gcp` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-gcp` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-gcp` module provides Terraform/OpenTofu components for GCP cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-gcp` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-huaweicloud module

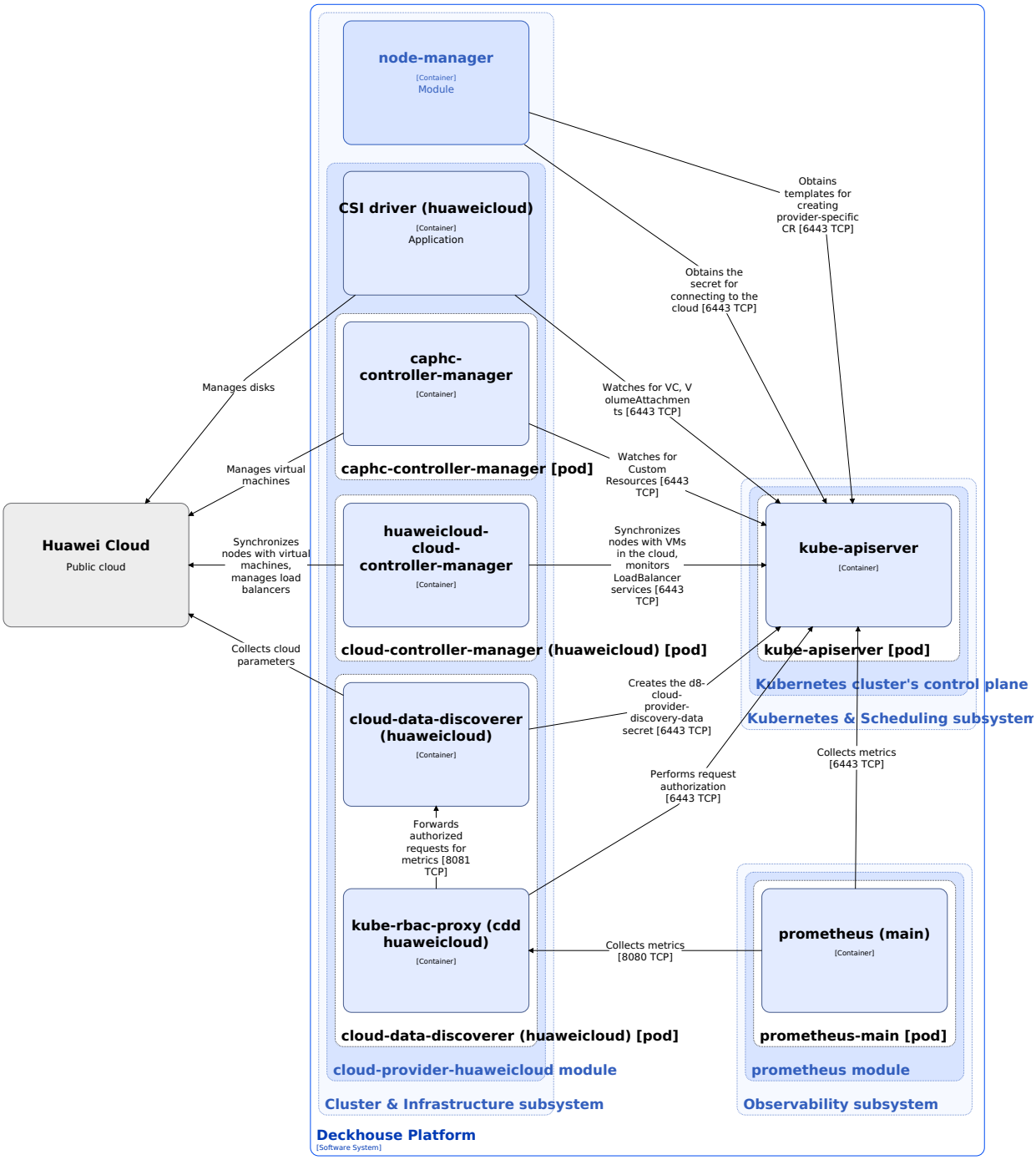
The `cloud-provider-huaweicloud` module is responsible for interacting with the Huawei Cloud resources. It allows the `node-manager` module to use Huawei Cloud resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-huaweicloud` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Caphc-controller-manager**: Kubernetes Cluster API Provider for Huawei Cloud. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, Huawei Cloud in this case. Caphc-controller-manager works with the following custom resources:

- HuaweiCloudCluster: Description of a Huawei Cloud-based cluster.
- HuaweiCloudMachineTemplate: Template describing the characteristics of the machines created in the cloud.
- HuaweiCloudMachine: Description of the characteristics of a machine created based on HuaweiCloudMachineTemplate.

It consists of a single container:

- **caphc-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Huawei Cloud. It provides interaction with the Huawei Cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **huaweicloud-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-huaweicloud` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (huaweicloud)**: It is an implementation of the CSI driver for Huawei Cloud. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (huaweicloud) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles HuaweiCloudCluster, HuaweiCloudMachineTemplate, and HuaweiCloudMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Huawei Cloud**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-huaweicloud` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-huaweicloud` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates

mentioned above.

2. The `cloud-provider-huaweicloud` module provides Terraform/OpenTofu components for Huawei Cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-huaweicloud` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-openstack module

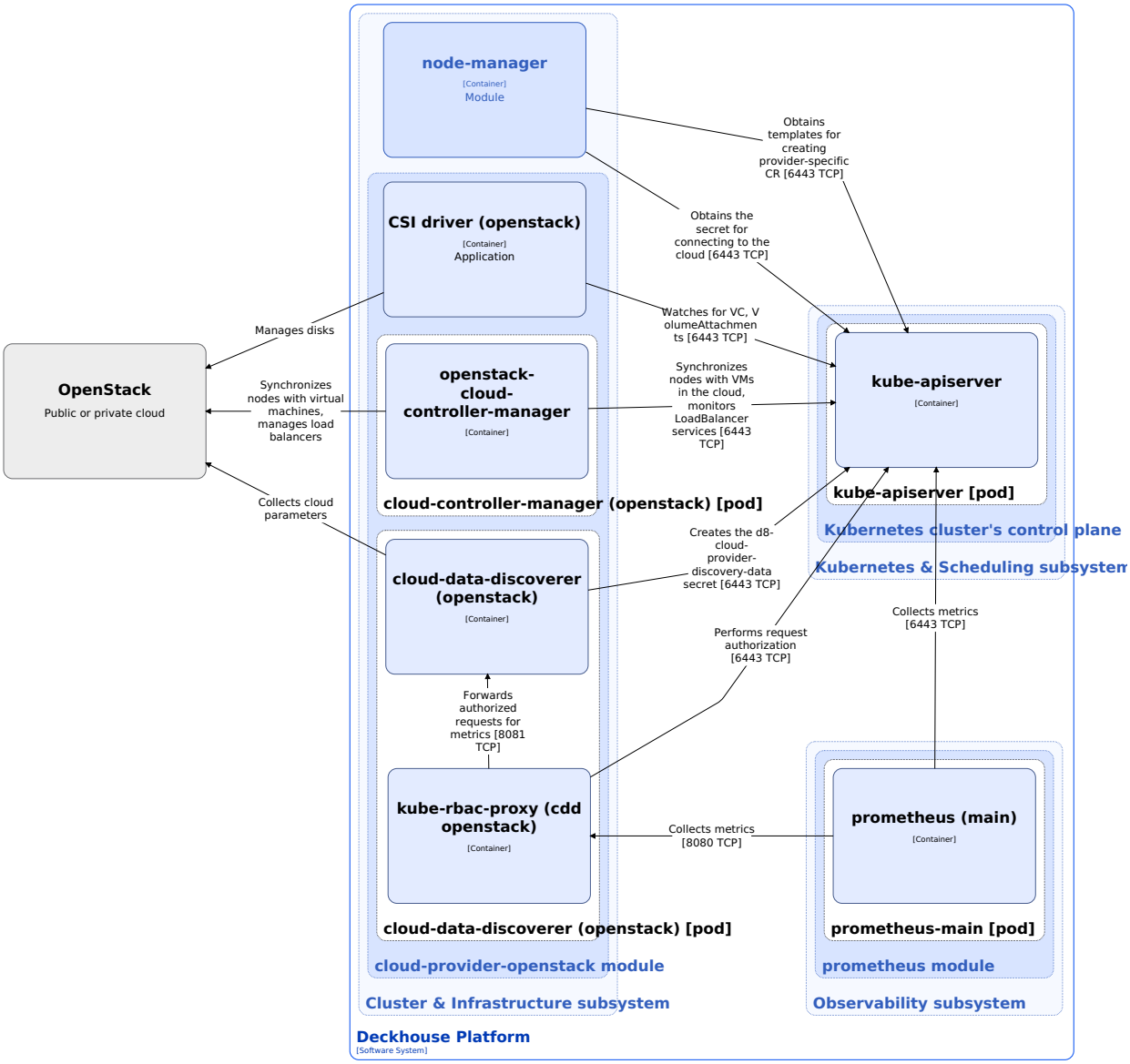
The `cloud-provider-openstack` module is responsible for interacting with the OpenStack cloud resources. It allows the `node-manager` module to use OpenStack resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-openstack` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: Kubernetes external cloud provider for OpenStack, an implementation of Cloud controller manager for OpenStack. It provides interaction with the OpenStack cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **openstack-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-openstack` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (openstack)**: It is an implementation of the CSI driver for OpenStack. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (openstack) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **OpenStack**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-openstack` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-openstack` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-openstack` module provides Terraform/OpenTofu components for OpenStack cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-openstack` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-vcd module

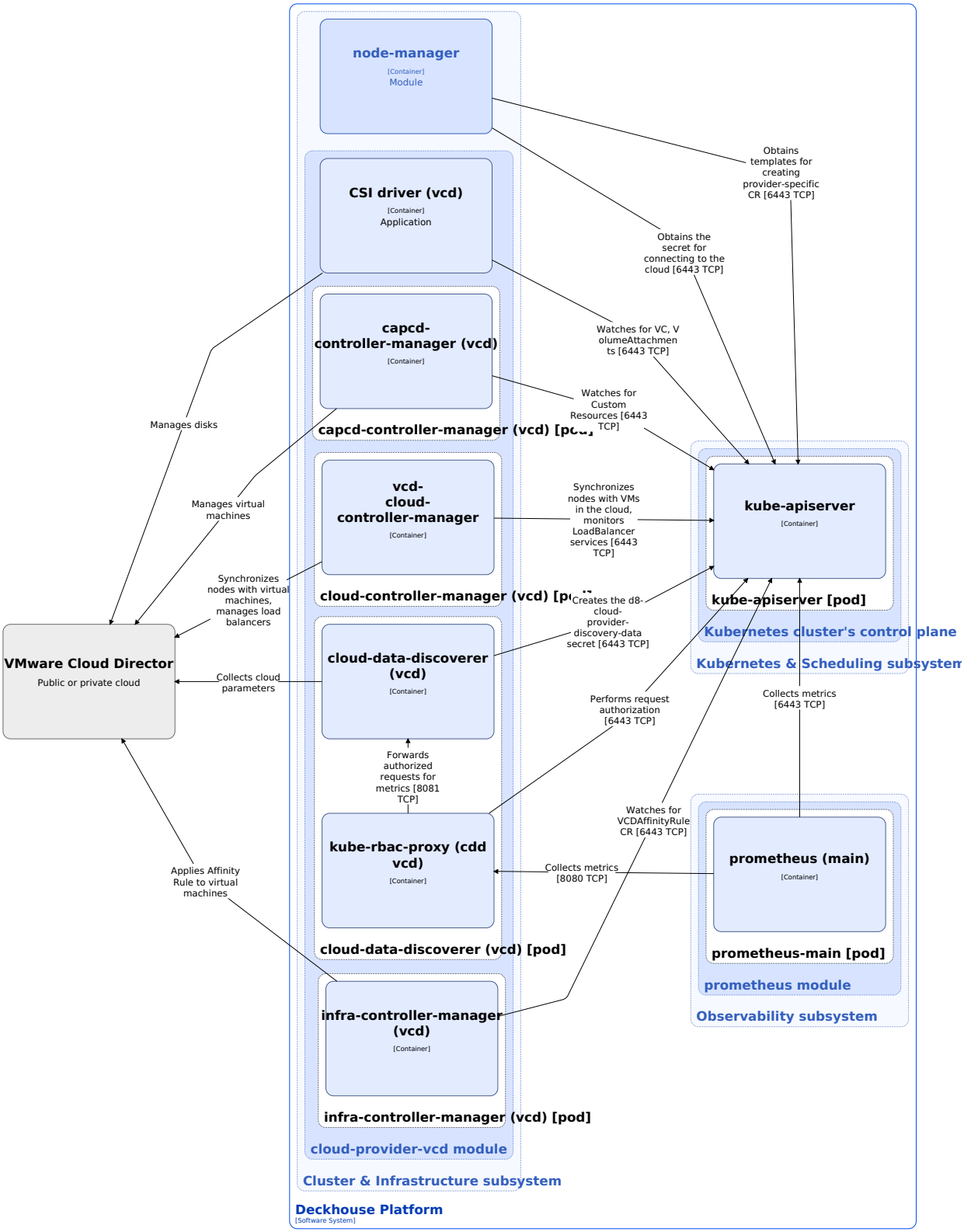
The `cloud-provider-vcd` module is responsible for interacting with the VMware Cloud Director cloud resources. It allows the `node-manager` module to use VMware Cloud Director resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-vcd` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capcd-controller-manager**: Kubernetes Cluster API Provider Cloud Director. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, VMware Cloud Director in this case. Capcd-controller-manager works with the following custom resources:
   - VCDClusterTemplate: Template describing the characteristics of the cluster created in the cloud.
   - VCDCluster: Description of the characteristics of a cluster created based on VCDClusterTemplate.
   - VCDMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - VCDMachine: Description of the characteristics of a machine created based on

VCDMachineTemplate.

It consists of a single container:

- **capcd-controller-manager**.

2. **Cloud-controller-manager**: Kubernetes External Cloud Provider for VMware Cloud Director. It is an implementation of cloud controller manager for VMware Cloud Director. It provides interaction with the VMware Cloud Director cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **vcd-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-vcd` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **Infra-controller-manager**: It is responsible for managing the schedulling of node location relative to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. Infra-controller-manager works with a custom resource VCDAffinityRule. It specifies the affinity rule applied only to virtual machines of node groups referencing this instance class within the cluster.

   It consists of a single container:

   - **infra-controller-manager**.

5. **CSI driver (VCD)**: It is an implementation of the CSI driver for VMware Cloud Director. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section. `Cloud-provider-vcd` module uses CSI driver for VMware Cloud Director Named Independent Disks.

   CSI driver (VCD) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles VCDClusterTemplate, VCDCluster, VCDMachineTemplate, VCDMachine, and VCDAffinityRule custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **VMware Cloud Director**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Applies Affinity Rules to virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-vcd` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-vcd` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-vcd` module provides Terraform/OpenTofu components for VMware Cloud Director cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-vcd` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-vsphere module

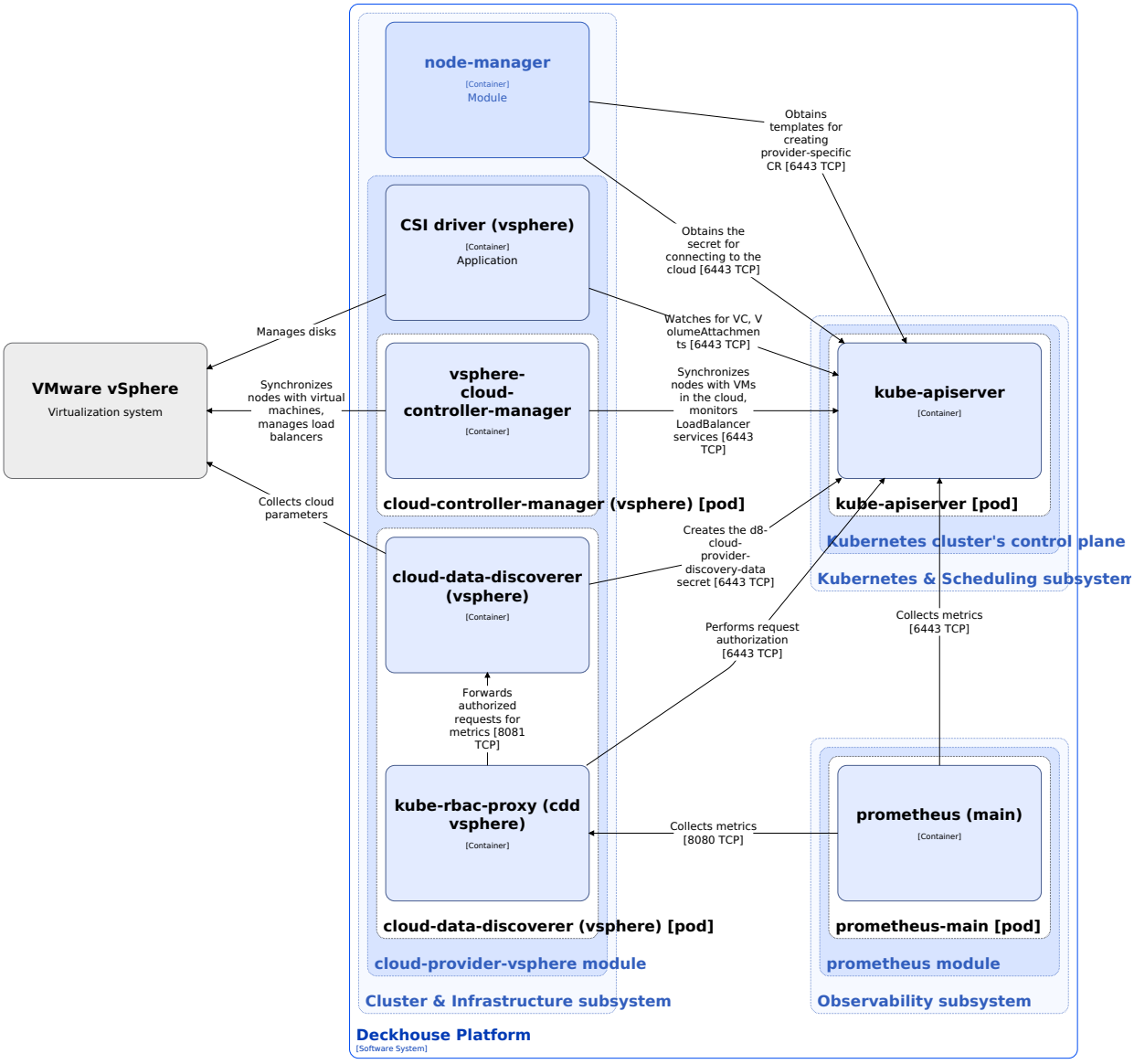
The `cloud-provider-vsphere` module is responsible for interacting with the VMware vSphere cloud resources. It allows the `node-manager` module to use VMware vSphere resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-vsphere` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: Kubernetes vSphere Cloud Provider. It is an implementation of cloud controller manager for VMware vSphere. It provides interaction with the VMware vSphere cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **vsphere-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-vsphere` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (vsphere)**: It is an implementation of the CSI driver for VMware vSphere. To study the `cloud-provider-vsphere` CSI driver architecture, refer to the corresponding documentation section.

   CSI driver (vsphere) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **VMware vSphere**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-vsphere` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-vsphere` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-vsphere` module provides Terraform/OpenTofu components for VMware vSphere cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-vsphere` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-yandex module

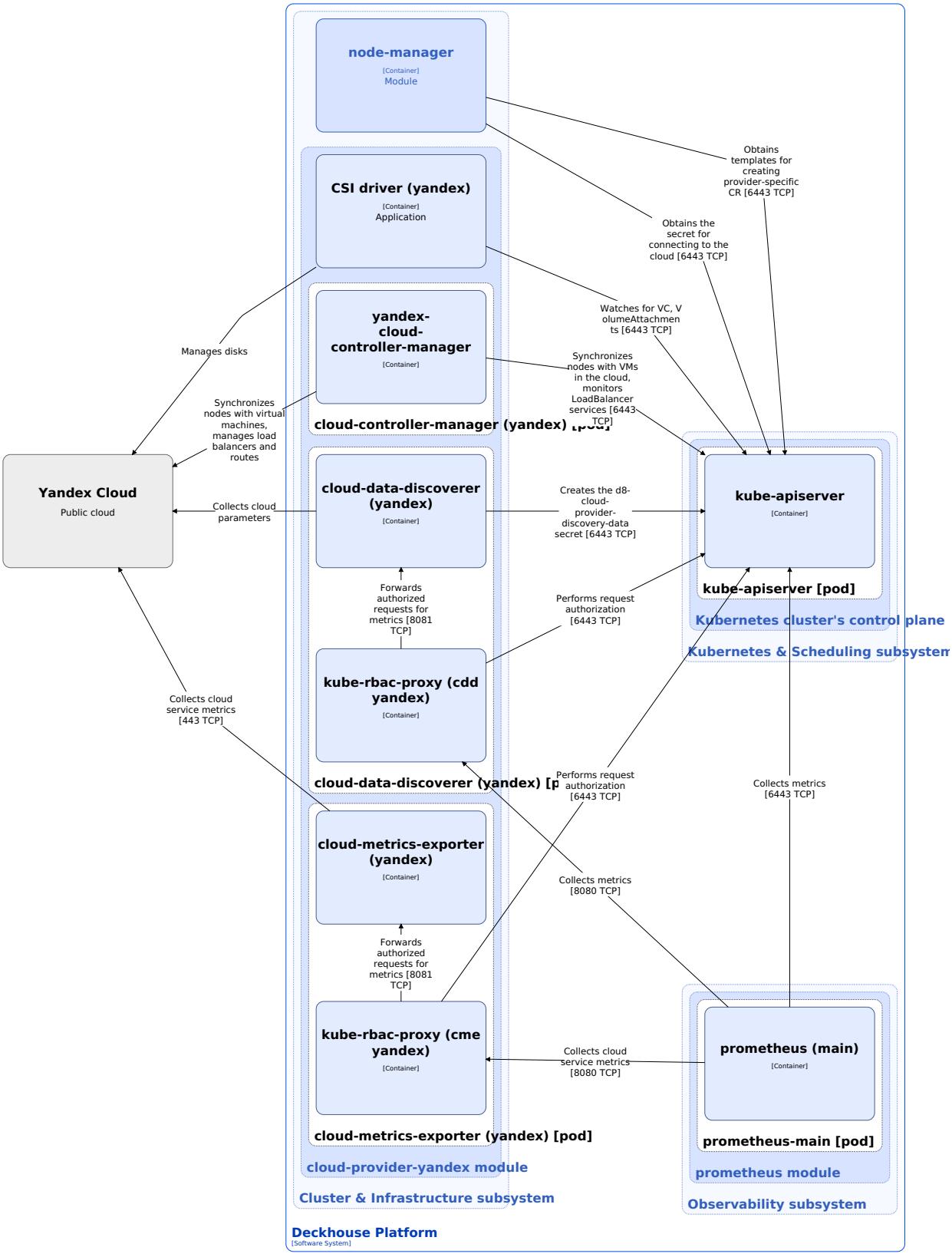
The `cloud-provider-yandex` module is responsible for interacting with the Yandex Cloud cloud resources. It allows the `node-manager` module to use Yandex Cloud resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-yandex` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Yandex Cloud. It provides interaction with the Yandex Cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

- Creates network routes for the `PodNetwork` network in the cloud.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **yandex-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-yandex` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **Cloud-metrics-exporter**: It is responsible for collecting metrics from the cloud provider's API and providing them in Prometheus format.

   It consists of the following containers:

   - **cloud-metrics-exporter**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-metrics-exporter metrics.

4. **CSI driver (yandex)**: It is an implementation of the CSI driver for Yandex Cloud. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (yandex) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.

2. **Yandex Cloud**:

   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.
   - Manages network routes for `PodNetwork` network.
   - Collects cloud service metrics.

The following external components interact with the module:

1. **Prometheus-main**:

   - Collects cloud-data-discoverer metrics.
   - Collects cloud provider's metrics using cloud-metrics-exporter.

Indirect interactions:

1. The `cloud-provider-yandex` module provides `node-manager` with following artifacts:

   - Provider-specific custom resource templates to be used by `cloud-provider-yandex` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.

2. The `cloud-provider-yandex` module provides Terraform/OpenTofu components for Yandex Cloud used when building the dhctl executable file for the `terraform-manager` module, such as:

   - Terraform/OpenTofu provider.
   - Terraform modules.

   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For

example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:

- `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
- `master-node`
- `static-node`

Additional resources

- Module `cloud-provider-yandex` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud-provider-zvirt module

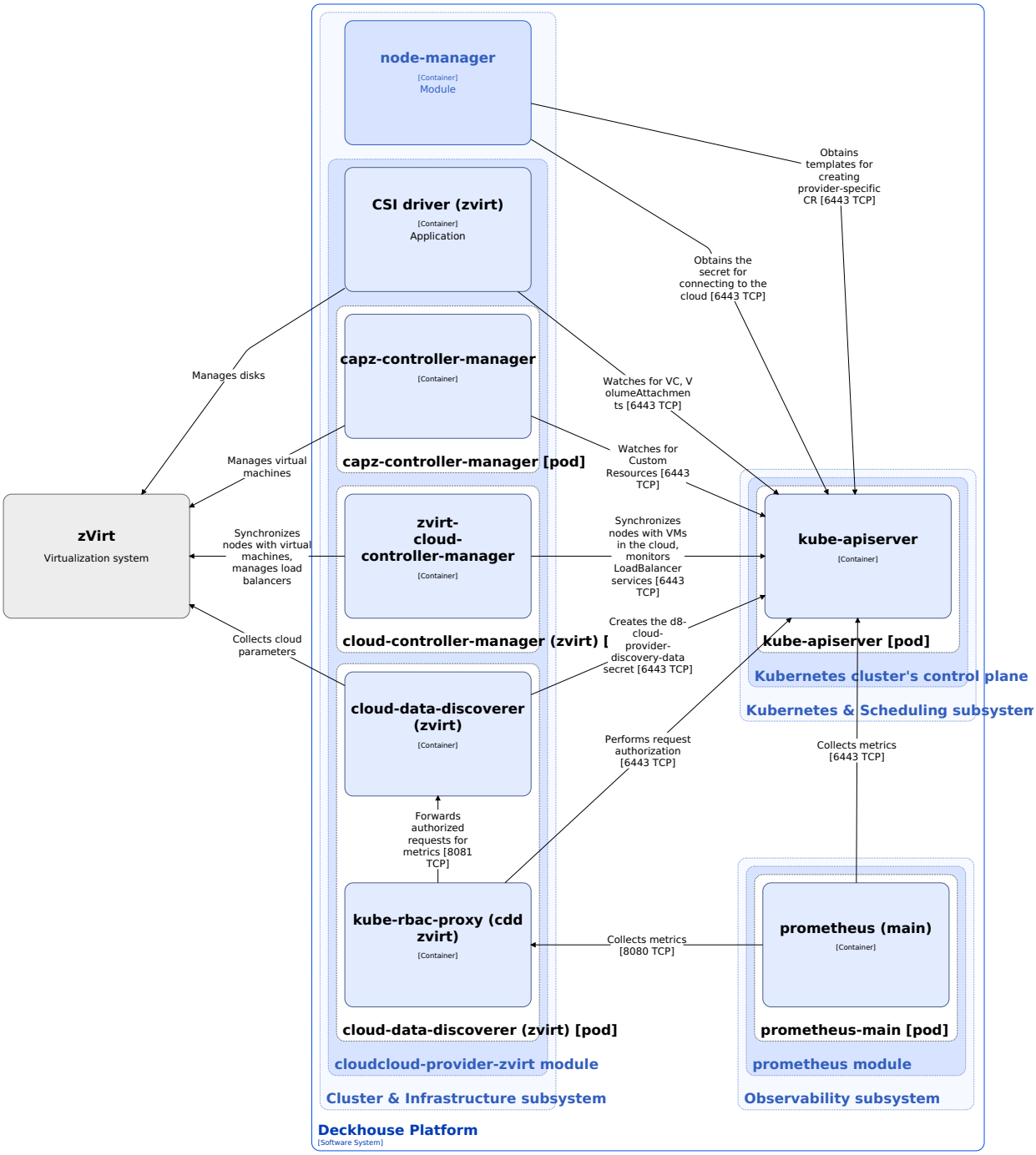
The `cloud-provider-zvirt` module is responsible for interacting with the zVirt cloud resources. It allows the `node-manager` module to use zVirt resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-zvirt` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capz-controller-manager**: Kubernetes Cluster API Provider for zVirt. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, zVirt in this case. Capz-controller-manager works with the following custom resources:

   - ZvirtCluster: Description of a zVirt-based cluster.
   - ZvirtMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - ZvirtMachine: Description of the characteristics of a machine created based on ZvirtMachineTemplate.

   It consists of a single container:

   - **capz-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for zVirt. It provides interaction with the zVirt cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud

provider. To do this:

- It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
- It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.

- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **zvirt-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-zvirt` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (zvirt)**: It is an implementation of the CSI driver for zVirt. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (zvirt) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles ZvirtCluster, ZvirtMachineTemplate, and ZvirtMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **zVirt**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-zvirt` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-zvirt` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-zvirt` module provides Terraform/OpenTofu components for zVirt cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`

- `static-node`

Additional resources

- Module `cloud-provider-zvirt` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Chrony module

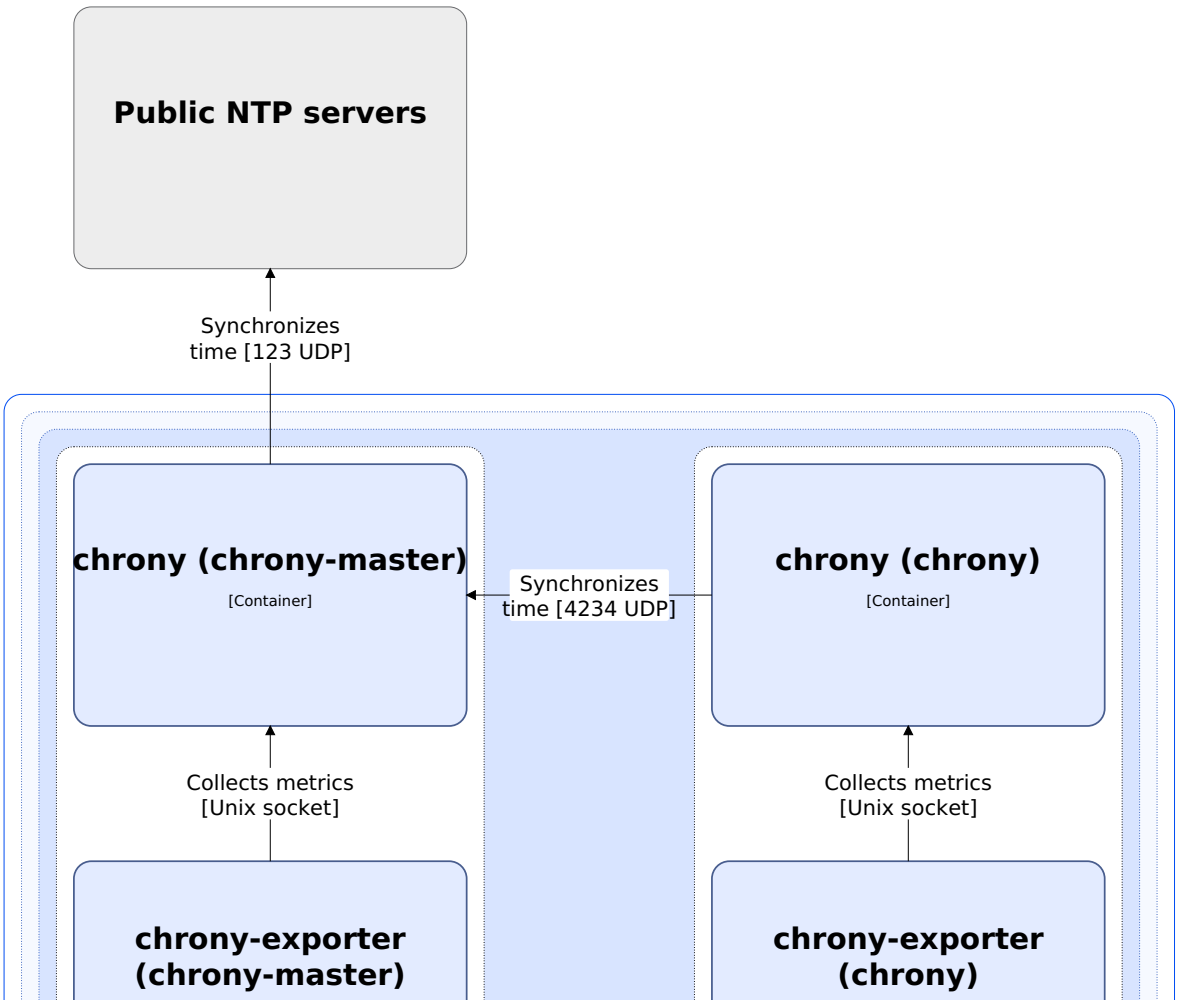
The `chrony` module provides time synchronization on all nodes in the Deckhouse Platform (DP) cluster using chrony NTP server/client implementation.

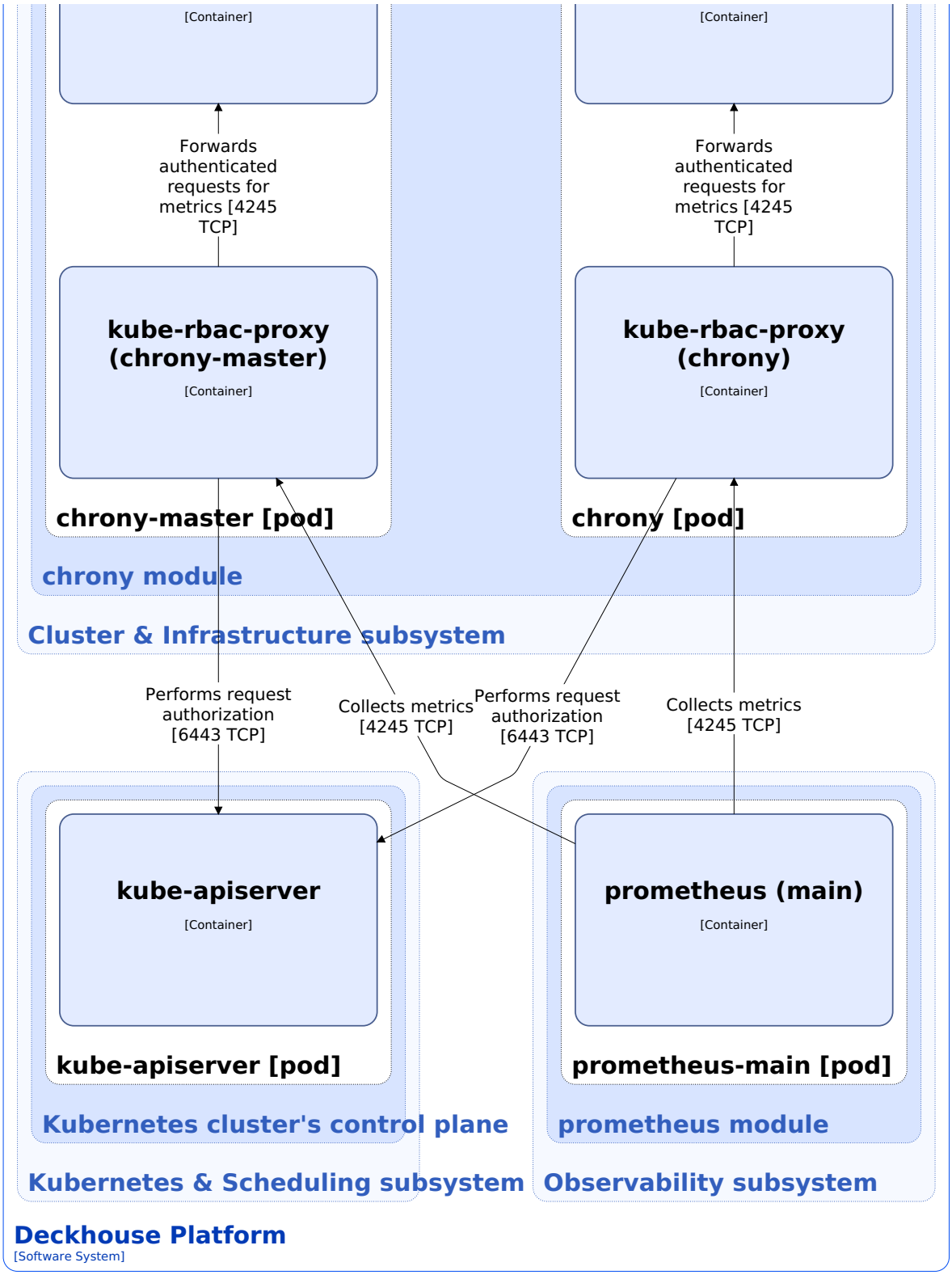
For more details, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `chrony` module its interactions with other DP components are shown in the following diagram:





Module components

The module consists of the following components:

1. **Chrony-master**: Time synchronization service on cluster master nodes with external NTP servers.

   It consists of the following containers:

   - **chrony**: Main container.
   - **chrony-exporter**: A sidecar container that collects metrics from the chrony container and exposes them in Prometheus format.
   - **kube-rbac-proxy**: A sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to metrics from the chrony-exporter container.

2. **Chrony**: Time synchronization service on all cluster nodes except master nodes, synchronized with the

chrony-master component.

It consists of the following containers:

- **chrony**: Main container.
- **chrony-exporter**: A sidecar container that collects metrics from the chrony container and exposes them in Prometheus format.
- **kube-rbac-proxy**: A sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to metrics from the chrony-exporter container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**: Authorizes requests to retrieve metrics.
2. **External NTP servers**: Performs time synchronization.

The following external components interact with the module:

- **Prometheus-main**: Collects metrics from chrony and chrony-master.

Additional resources

- Module `chrony` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Gpu module

The `gpu` module manages Graphics Processing Units (GPU) in Deckhouse Platform.

The module operates in two mutually exclusive modes. The mode is defined by the `dra.enabled` parameter:

- Dynamic Resource Allocation (DRA): The module uses a Kubernetes mechanism for requesting and sharing devices that provides dynamic and declarative allocation of GPU compute resources.
- Device Plugin mode (default): The module uses the classic Kubernetes model for working with node compute resources and publishes `nvidia.com/gpu` or `nvidia.com/mig-*` resources, which kube-scheduler uses to schedule pods that consume these resources.

For more details, refer to the corresponding documentation section.

The module architecture depends on the operating mode.

DRA mode

In DRA mode, the module is built around a vendor-independent core and adapters for the NVIDIA and MetaX vendors.

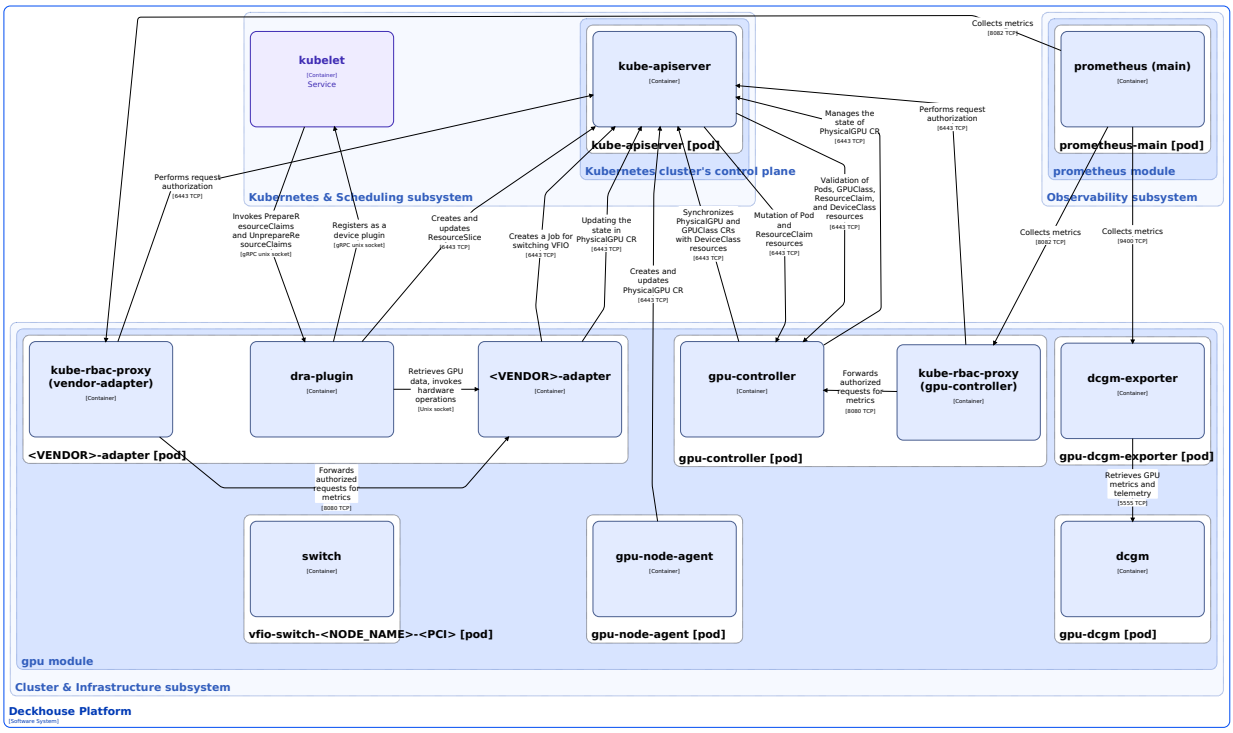
The module works with the following resources:

- DeviceClass: A DRA resource that stores descriptions of device classes that can be used for dynamic resource assignment.
- GPUClass: A custom resource that stores requirements for a GPU group (memory size, hardware capabilities) and the usage and compatibility policy.
- PhysicalGPU: A custom resource that stores a description of a physical GPU, including device characteristics.
- ResourceClaim: A DRA resource that contains a request to allocate a resource for a pod and describes the required characteristics and usage parameters.
- ResourceSlice: A DRA resource that represents an allocated share or part of a resource assigned within a ResourceClaim.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `gpu` module in DRA mode and its interactions with other Deckhouse Platform components are shown in the following diagram:



Module components

The module consists of the following components:

1. **gpu-controller** (Deployment): A controller that processes GPU resource requests and runs admission webhooks for DRA objects through the Validating/Mutating Admission Controllers mechanism. The controller runs on master nodes.

   The gpu-controller performs the following actions:

   - Creates and updates DeviceClass DRA resources based on PhysicalGPU and GPUClass custom resources.
   - Validates and mutates Pod create and update requests, creating ResourceClaim resources when necessary.
   - Watches ResourceClaim resource changes and, based on this data, maintains the state and occupancy of PhysicalGPU resources.
   - Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
   - Manages the PhysicalGPU state.

   It consists of the following containers:

   - **gpu-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container. This component is an Open Source project.

2. **gpu-node-agent** (DaemonSet): A component that consists of a single **gpu-node-agent** container and performs the following actions:

   - Scans the host `/sys` filesystem and the PCI ID database.
   - Matches devices against the `gpu-supported-vendors` ConfigMap.
   - Creates PhysicalGPU custom resources for each discovered GPU.
   - Sets the `gpu.deckhouse.io/vendor=<VENDOR>` labels on the Node resource.

   The component runs on all cluster nodes except the control plane.

3. **<VENDOR>-adapter** (DaemonSet): A component that works with hardware to prepare, allocate, and

release GPU resources. Currently, two hardware vendors are supported: NVIDIA and MetaX.

The component performs the following actions:

- Registers in kubelet as a DRA kubelet plugin.
- Prepares and releases allocated resources for pods through the `PrepareResourceClaims` and `UnprepareResourceClaims` operations.
- Publishes the list of available devices through ResourceSlice resources.
- Retrieves hardware capabilities.
- Partitions GPU resources and makes them available to pods.
- Enriches the status in PhysicalGPU resources.

It consists of the following containers:

- **dra-plugin**: Sidecar container that implements the DRA kubelet plugin and interacts with the vendor adapter.
- **<VENDOR>-adapter**: Vendor-specific sidecar container that interacts with the hardware on the node.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the dra-plugin container.

The component runs on all cluster nodes that have the `gpu.deckhouse.io/vendor=<VENDOR>` label.

4. **gpu-dcgm** (DaemonSet): A component that consists of a single **dcgm** container and runs Data Center GPU Manager (DCGM). DCGM collects GPU health and utilization data (Error Correction Code (ECC), power). It works only with NVIDIA cards.
5. **gpu-dcgm-exporter** (DaemonSet): A component that consists of a single **dcgm-exporter** container, retrieves GPU metrics from the gpu-dcgm component, and exposes them in Prometheus format.
6. **vfio-switch-<NODE_NAME>-<PCI>** (Job): A component that consists of a single **switch** container. It switches the driver in use between `nvidia` and `vfio-pci`. The component is created by nvidia-adapter to control the switching process.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests to retrieve metrics.
   - Works with PhysicalGPU and GPUClass custom resources.
   - Updates Node resources.
   - Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
   - Works with DeviceClass, ResourceClaim, and ResourceSlice resources.
   - Creates and monitors `vfio-switch-<NODE_NAME>-<PCI>` Jobs.
2. **Kubelet**: Registers the module as a DRA kubelet plugin.

The following external components interact with the module:

1. **Kubelet**: Calls the `PrepareResourceClaims` and `UnprepareResourceClaims` gRPC methods.
2. **Kube-apiserver**: Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
3. **Prometheus-main**: Collects metrics from the gpu-dcgm and <VENDOR>-adapter components.

Device Plugin mode

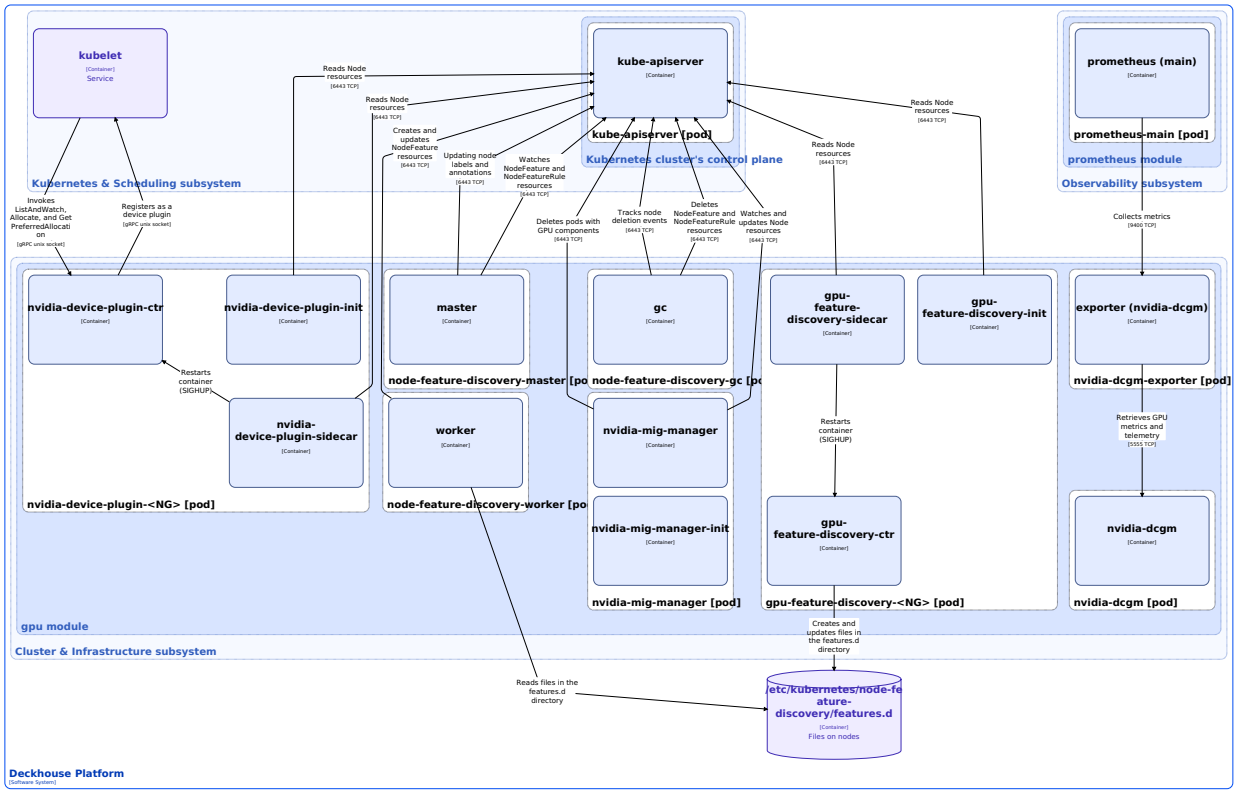
In Device Plugin mode, the module consists of components that work only with NVIDIA adapters.

The module interacts with the following resources:

- NodeFeature: Stores actual information about the hardware capabilities of a specific node.
- NodeFeatureRule: Stores a set of rules used by the module to configure labels, annotations, and taints for a cluster node.

Module architecture

The Level 2 C4 architecture of the `gpu` module in Device Plugin mode and its interactions with other Deckhouse Platform components are shown in the following diagram:



Module components

The module consists of the following components:

1. **node-feature-discovery-master** (Deployment): A component that consists of a single **master** container, which collects information about node hardware capabilities from NodeFeature resources, and publishes them as `feature.node.kubernetes.io/*` and `nvidia.com/*` labels on the corresponding nodes. The master obtains the rules for assigning labels, taints, and annotations from NodeFeatureRule resources.

2. **node-feature-discovery-worker** (DaemonSet): A component that consists of a single **worker** container, which runs on each GPU node, discovers connected PCI and USB devices on the node, and publishes them as NodeFeature resources. The component also publishes information received from the gpu-feature-discovery-<NG> component as NodeFeature resources.

3. **node-feature-discovery-gc** (Deployment): A component that consists of a single **gc** container, which deletes obsolete NodeFeature resources when a node is deleted.

4. **gpu-feature-discovery-<NG>** (DaemonSet): A component that queries the GPU driver via the NVIDIA Management Library (NVML) and writes information about GPU hardware capabilities to the `/etc/kubernetes/node-feature-discovery/features.d/gfd` file. Node-feature-discovery-worker publishes this information as NodeFeature resources, from which node-feature-discovery-master updates the corresponding `nvidia.com/*` labels for cluster nodes.

   The module creates the component for every NodeGroup (NG) whose configuration specifies the `.spec.gpu` parameter.

   It consists of the following containers:

   - **gpu-feature-discovery-init**: Init container that prepares the configuration for the main gpu-feature-discovery-ctr container.
   - **gpu-feature-discovery-ctr**: Main container.
   - **gpu-feature-discovery-sidecar**: Sidecar container that watches configuration changes and restarts the main container to apply them.

5. **nvidia-device-plugin-<NG>** (DaemonSet): A component that registers with kubelet through the Kubernetes Device Plugin API and publishes GPU resources for kube-scheduler.

   To work with resources, kubelet calls the `ListAndWatch`, `Allocate`, and `GetPreferredAllocation` gRPC methods on nvidia-device-plugin-ctr. After that, the component updates the number of available resources on the node and returns this information through kubelet.

   It consists of the following containers:

   - **nvidia-device-plugin-init**: Init container that prepares the configuration for the main nvidia-device-plugin-ctr container.

    - **nvidia-device-plugin-ctr**: Main container.
    - **nvidia-device-plugin-sidecar**: Sidecar container that watches configuration changes and restarts the main container to apply them.

6. **nvidia-mig-manager** (DaemonSet): An optional component that manages changes to the Multi-Instance GPU (MIG) profile on nodes with GPUs A100 and H100.

   The component performs the following actions:

    - Retrieves the desired MIG profile (the `nvidia.com/mig.config` label) and the current state.
    - Puts the node into maintenance mode when necessary (applies a taint, cordon, and drain).
    - Stops pods that use GPUs on the node.
    - Applies the MIG profile.
    - Initiates a node reboot when necessary.
    - Returns the node to service.

   It consists of the following containers:

    - **nvidia-mig-manager-init**: Init container that prepares executables and libraries.
    - **nvidia-mig-manager**: Main container.

7. **nvidia-dcgm** (DaemonSet): A component that consists of a single **nvidia-dcgm** container, which runs Data Center GPU Manager (DCGM). DCGM collects GPU health and utilization data (Error Correction Code (ECC), power).

8. **nvidia-dcgm-exporter** (DaemonSet): A component that consists of a single **exporter** container, which retrieves GPU metrics from the nvidia-dcgm component, and exposes them in Prometheus format.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Authorizes requests to retrieve metrics.
    - Works with NodeFeature and NodeFeatureRule resources.
    - Watches and updates Node resources.
    - Terminates pods that use GPU resources when the MIG profile changes.

2. **Kubelet**: Registers the module through the Device Plugin API.

The following external components interact with the module:

1. **Kubelet**: Calls the `ListAndWatch`, `Allocate`, and `GetPreferredAllocation` gRPC methods.

2. **Prometheus-main**: Collects metrics from nvidia-dcgm.

Additional resources

- Module `gpu` documentation
- Module `node-manager` custom resources

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Registry-packages-proxy module

The `registry-packages-proxy` module provides an in-cluster HTTP proxy service for accessing packages from container registries in the Deckhouse Platform (DP). It acts as an intermediary between cluster components and external or internal registries, offering caching capabilities to optimize bandwidth usage and improve package retrieval performance.

This module is a critical infrastructure component that runs on master nodes and is used during cluster bootstrap, as well as during cluster operation, to fetch packages from container registries.

The module deploys a highly-available proxy service that:

- Runs on master nodes with `hostNetwork` enabled to ensure availability during bootstrap when CNI is not yet available.
- Exposes a separate HTTP endpoint on port `4282` for downloading `rpp-get` (a utility for interacting with the module's main endpoint) during node bootstrap. Requests to this port do not go through TLS or

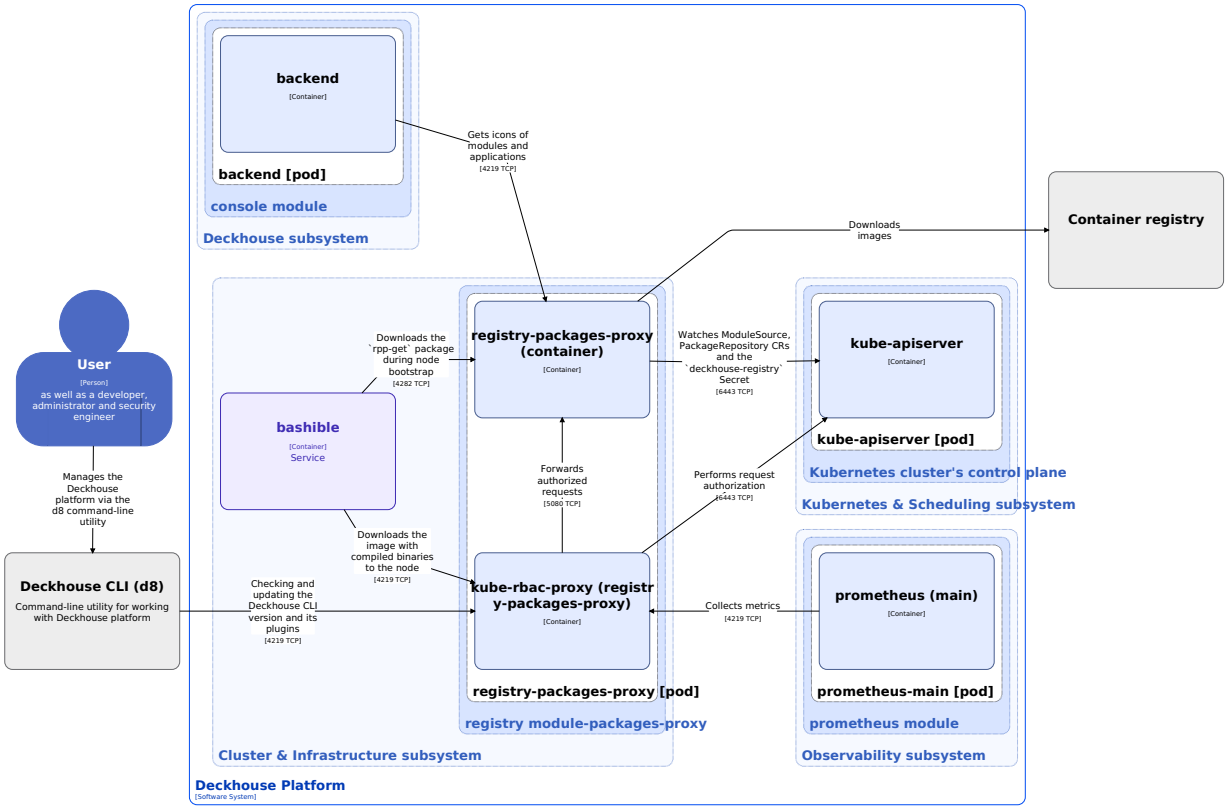
kube-rbac-proxy, unlike the main proxy on port `4219`.
- Listens on port `4219` (HTTPS) on each master node's IP address.
- Provides a `GET /package` endpoint for retrieving registry packages by digest.
- Implements local caching of retrieved packages (up to 1 GB) to reduce network traffic and improve performance.
- Watches the `deckhouse-registry` Secret in the `d8-system` namespace to obtain credentials for the main registry.
- Watches ModuleSource and PackageRepository custom resources to obtain registry credentials.
- Uses RBAC-based authorization to secure access to the proxy and metrics endpoints.
- Exposes a public HTTPS API (via Ingress) for Deckhouse CLI binaries and plugins.
- Exposes an in-cluster HTTPS API for package icons (no public Ingress).

For more information about the module, see the module overview section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `registry-packages-proxy` module and its interactions with other DP components are shown in the following diagram:



Module components

The `registry-packages-proxy` module consists of a single **registry-packages-proxy** component that includes the following containers:

- **registry-packages-proxy**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to the endpoints of the registry-packages-proxy main container. This component is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Reading and watching ModuleSource and PackageRepository custom resources.
    - Watching the `deckhouse-registry` Secret.
    - Authorizing requests to access the endpoints of the main container.

2. **Container registry**: Downloads container images from container registries.

The following external components interact with the module:

1. **Bashible**: Downloads the image with assembled binaries to nodes during cluster bootstrap.
2. **Prometheus-main**: Collects metrics from the registry-packages-proxy container.
3. **Console**: Retrieves package icons.
4. **Deckhouse CLI**: Retrieves versions of the `d8` utility and its plugins.

Additional resources

- Global custom resources
- Module `registry-packages-proxy` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Terraform-manager module

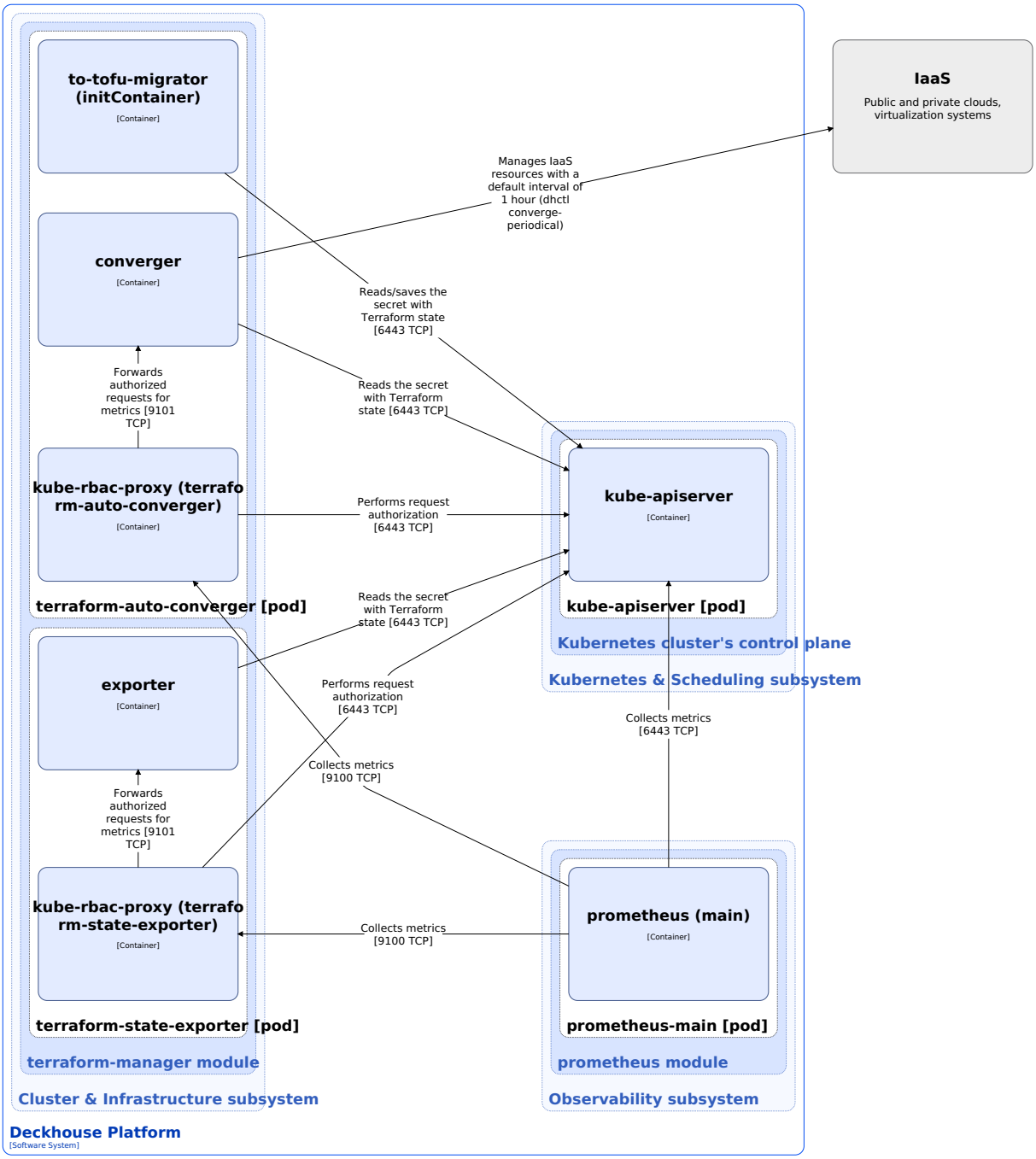
The `terraform-manager` module provides tools for managing the Terraform state in a DP cluster.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `terraform-manager` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Terraform-auto-converger**: Periodically (once per hour by default) checks the Terraform state and applies non-destructive changes to infrastructure resources.

   The component operates only on the base infrastructure of the cluster. Cluster nodes are not automatically reconciled to the desired state. The check interval is configured using the `autoConvergerPeriod` parameter.

   It consists of the following containers:

   - **to-tofu-migrator**: Init container used to migrate the Terraform state to OpenTofu. It runs the `dhctl` utility with the `converge-migration` command.
   - **converger**: Main container running the `dhctl` utility with the `converge-periodical` command.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the metrics of the converger container. This component is an open-source project.

2. **Terraform-state-exporter**: Checks the Terraform state and exports related metrics.

   It consists of the following containers:

   - **exporter**: Main container running the `dhctl` utility with the `terraform converge-exporter` command.

- **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the metrics of the exporter container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Reading and writing the Secret containing the Terraform state.
   - Authorization of metric requests.
2. **Cloud infrastructure** (or virtualization platform): Manages base infrastructure resources and reconciles them to the desired state.

The following external components interact with the module:

- **prometheus-main**: Collects metrics from terraform-auto-converger and terraform-state-exporter.

Additional resources

- Module `terraform-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

IAM subsystem

This subsection describes the architecture of the IAM subsystem (Identity and Access Management) of Deckhouse Platform (DP).

The IAM subsystem provides the following features in DP:

- User authentication
- Role-based access control (RBAC)
- Multitenancy
- Automatic assignment of annotations and labels to namespaces

The IAM subsystem includes the following modules that implement the features described above:

- `user-authn`: User authentication.
- `user-authz`: Role-based access control (RBAC).
- `multitenancy-manager`: Multitenancy.
- `namespace-configurator`: Automatic assignment of annotations and labels to namespaces.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Authentication

Connecting to Kubernetes API using a generated kubeconfig

1. **Initialization**. kube-apiserver reads the configuration of the OIDC provider (in this case — Dex): the `issuer` and the JWKS endpoint for token validation. In Kubernetes 1.30 and later, it reads it through the Dex Service inside the cluster.
2. **Kubeconfig generation**. The Deckhouse Platform (DP) web UI generates a kubeconfig file that includes an `ID token` and a `refresh token`. This file is used by `kubectl` or other Kubernetes clients.
3. **Authentication when accessing the API**. Upon receiving a request with an `ID token`, the `kube-apiserver` verifies the token's signature using keys from the JWKS endpoint. It then compares the `iss` (issuer) and `aud` (audience) claims in the token against the server configuration.

Protection against credential brute-forcing in Dex

Dex limits failed sign-in attempts per client IP address on the password endpoints. Configure the limit with the `rateLimit` parameter. It is turned off by default.

Dex also locks an account after failed sign-in attempts. Configure the lockout with the `passwordPolicy.lockout` parameter. It applies to local users, and to LDAP and Crowd users if their types are listed in `applyToConnectors`.

Authentication via DexAuthenticator

1. **Login process via Dex**. In most cases, Dex redirects the user to the login page of an external identity provider (e.g., GitHub, Okta, Keycloak), then expects the user to return to the `/callback` URL after successful authentication.
   However, for providers like LDAP or Atlassian Crowd, this flow is not supported. Instead, the user enters their login and password in the Dex login form, and Dex performs authentication by calling the provider's API directly.
2. **Token and session storage**. DexAuthenticator sets a cookie containing the full `refresh token`, rather than issuing a short-lived ticket as with the `ID token`.
   This is because Redis, used by DexAuthenticator, does not persist data to disk.
   If the `ID token` is missing from Redis, the user can obtain a new one using the `refresh token` stored in the cookie.
3. **Passing the token to the application**. DexAuthenticator sets the `Authorization` HTTP header with the `ID token` from Redis.
   This may be optional for some applications, such as Upmeter, where other authorization mechanisms are used.
   However, for applications like the Kubernetes Dashboard, this behavior is critical, as the Dashboard forwards the `ID token` to access the Kubernetes API on behalf of the user.

DP-specific Dex extensions

DP ships Dex with additional capabilities:

- Groups for static users (the Group object).
- Two-factor authentication (2FA) for local users and for the users of LDAP and Atlassian Crowd providers (the `twoFactor` parameter).
- A password policy for static users: complexity, password history, rotation and lockout after failed sign-in attempts (the `passwordPolicy` parameter).
- A forced password change at the next sign-in after an administrator resets the password.
- Lockout after failed sign-in attempts for LDAP and Atlassian Crowd users (the `passwordPolicy.lockout.applyToConnectors` parameter).
- Per-IP rate limiting of failed sign-in attempts (the `rateLimit` parameter).
- Locking and unlocking users with the UserOperation object, also without a password policy.
- Lists of allowed users and groups in DexClient and DexAuthenticator that Dex checks at sign-in and at every refresh-token request.
- Limits on the identities a provider may assert (the `spec.allowedIdentities` parameter of DexProvider).
- Kerberos (SPNEGO) sign-in through the LDAP provider without the password form, with tickets from the `Authorization: Negotiate` header validated by the service keytab.
- Refresh tokens for the SAML provider.
- A custom CA certificate for the GitLab provider (the `gitlab.rootCAData` parameter).

High availability mode

In high availability mode (`highAvailability`), Dex and the other `user-authn` components run in several replicas, so sign-in keeps working if one pod fails. DexAuthenticator follows the same setting unless its own `spec.highAvailability` is set.

Additional resources

- Module `user-authn` parameters
- Module `user-authn` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Multitenancy

The `multitenancy-manager` module allows you to create isolated projects within the Deckhouse Platform (DP). Projects provide resource quotas, network isolation, and security features that go beyond standard namespaces.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Internal logic

Project creation

The following custom resources are used to create a project:

- ProjectTemplate: Describes the project template. It defines the list of resources to be created in the project and the parameter schema that can be passed during project creation.
- Project: Describes a specific project.

When a Project is created from a specified ProjectTemplate, the following occurs:

1. The provided parameters are validated against the OpenAPI specification (defined in the `parametersSchema.openAPIV3Schema` parameter of the ProjectTemplate resource).
2. A namespace is created with the name of the Project.
3. The project objects are created from the template. In a `deckhouse.io/v1alpha2` template, settings such as the pod security profile, network isolation, log collection and node placement are described by structured fields, and the controller creates the corresponding objects in every namespace of the project. Other objects are described in `manifests` and are applied with the rights of the `d8:project:admin` role in the project namespaces, not with the rights of the controller. A template with a Helm text in `resourcesTemplate` is rendered through Helm. This form is deprecated, no template can get a new `resourcesTemplate`, and a template is created through `deckhouse.io/v1alpha2` only.
4. The standard fields of the project are applied independently of the template: `.spec.quota` is reconciled into a ResourceQuota, and `.spec.administrators` into an auto-managed ProjectRoleBinding.

> ⚠ If you modify a project template, all previously created projects will be updated according to the new template.

Project isolation

A namespace alone does not isolate a project. The project adds the following:

- Access to the whole project through `.spec.administrators`, ProjectRoleBinding and ClusterProjectRoleBinding. A plain RoleBinding in one namespace is allowed for delegatable roles.
- A resource limit through `.spec.quota`, applied as the `d8-project-quota` ResourceQuota in the main namespace.
- Network isolation through the `networkPolicy.mode: Isolated` field of the template.
- The pod security profile through the `podSecurityStandard` field of the template.

Managing access to cluster-wide resources

Objects in project namespaces can reference cluster-wide resources. For example, a PersistentVolumeClaim can use a StorageClass, a Certificate can use a ClusterIssuer, and a RoleBinding can use a ClusterRole. The module allows cluster administrators to define which cluster-wide resources can be used from project namespaces and which values are used by default.

This mechanism works independently of RBAC. RBAC determines *who can create and modify* objects, while the cluster-wide resource access mechanism determines *which resources* those objects can use.

The mechanism uses four custom resources:

- GrantableClusterResourceDefinition registers a type of cluster-wide resource whose access can be managed. These resources are provided by DP or module developers.
- GrantableClusterResourceReference defines where a registered cluster-wide resource is used, for example, which resource field contains a reference to it. These resources are provided by modules.
- ClusterResourceGrantPolicy defines access rules. In the `v1alpha2` version, a cluster administrator selects the projects to which the policy applies by the labels of Project objects (`projectSelector`) and narrows the choice by the labels of project namespaces (`namespaceSelector`). The policy also defines the allowed and denied resources, as well as the resource used by default.
- In every namespace of every project, the controller keeps an AvailableClusterResource for each GrantableClusterResourceDefinition. It lists the cluster-wide resources available to the project under the policies that apply and is read-only.

Until the administrator creates a ClusterResourceGrantPolicy, resource availability is determined by their registration: resources are available to all projects if `defaultAvailability: All` (the default value) is set in GrantableClusterResourceDefinition and the resource does not match the `excluded` or `reserved` filters.

Access checks are performed only for objects in project namespaces. If an access policy changes, cluster-wide resources already used by existing objects remain available to those objects.

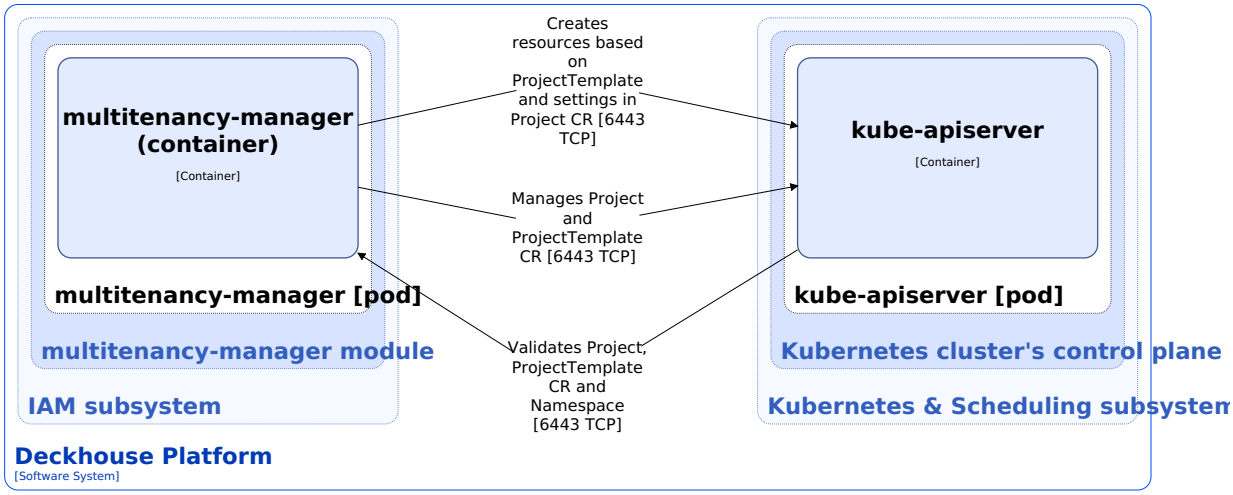
The webhooks reject requests while the module controller is unavailable. Until it is back, objects of the registered kinds (for example, PersistentVolumeClaim, Service or RoleBinding) cannot be created or changed in project namespaces, except by the platform components and members of the `system:masters` group. To keep the controller available, enable `highAvailability`.

For a detailed description of the access management mechanism, refer to the `multitenancy-manager` module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The architecture of the `multitenancy-manager` module at Level 2 of the C4 model and its interactions with other DP components are shown in the following diagram.



Module components

The module consists of the following components:

- **Multitenancy-manager**: The component consists of a single **multitenancy-manager** container and provides the following functions:
  - Managing the Project and ProjectTemplate custom resources.
  - Validating the Project and ProjectTemplate custom resources.
  - Creating the resources specified in the ProjectTemplate custom resource based on the parameters set in Project.
  - Managing the grants mechanism for cluster-wide resources — the `GrantableClusterResourceDefinition`, `GrantableClusterResourceReference`, `ClusterResourceGrantPolicy`, and `AvailableClusterResource` custom resources. It renders the per-project AvailableClusterResource catalogs and runs the validating and mutating admission webhooks (`/is-granted`, `/defaults`, `/protect`) that enforce which cluster resources a project may reference and substitute per-project defaults on CREATE.

> **i** **Warning.** Multitenancy-manager has `cluster-admin` permissions. Objects described in the `manifests` field of ProjectTemplate are applied not with these permissions but with the rights of the `d8:project:admin` role in the project namespaces.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Managing the Project and ProjectTemplate custom resources.
  - Validating the Project and ProjectTemplate custom resources.
  - Creating the resources specified in the ProjectTemplate custom resource based on the parameters set in Project.
  - Managing the GrantableClusterResourceDefinition, GrantableClusterResourceReference, ClusterResourceGrantPolicy, and AvailableClusterResource custom resources.
  - Validating the AvailableClusterResource custom resources.
  - Creating validation admission webhooks based on conditions in GrantableClusterResourceDefinition and GrantableClusterResourceReference custom resources.

Additional resources

- Module `multitenancy-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

User-authn module

The `user-authn` module implements a unified authentication system integrated with Kubernetes and the web interfaces used by modules of Deckhouse Platform (DP), such as the `console` module.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

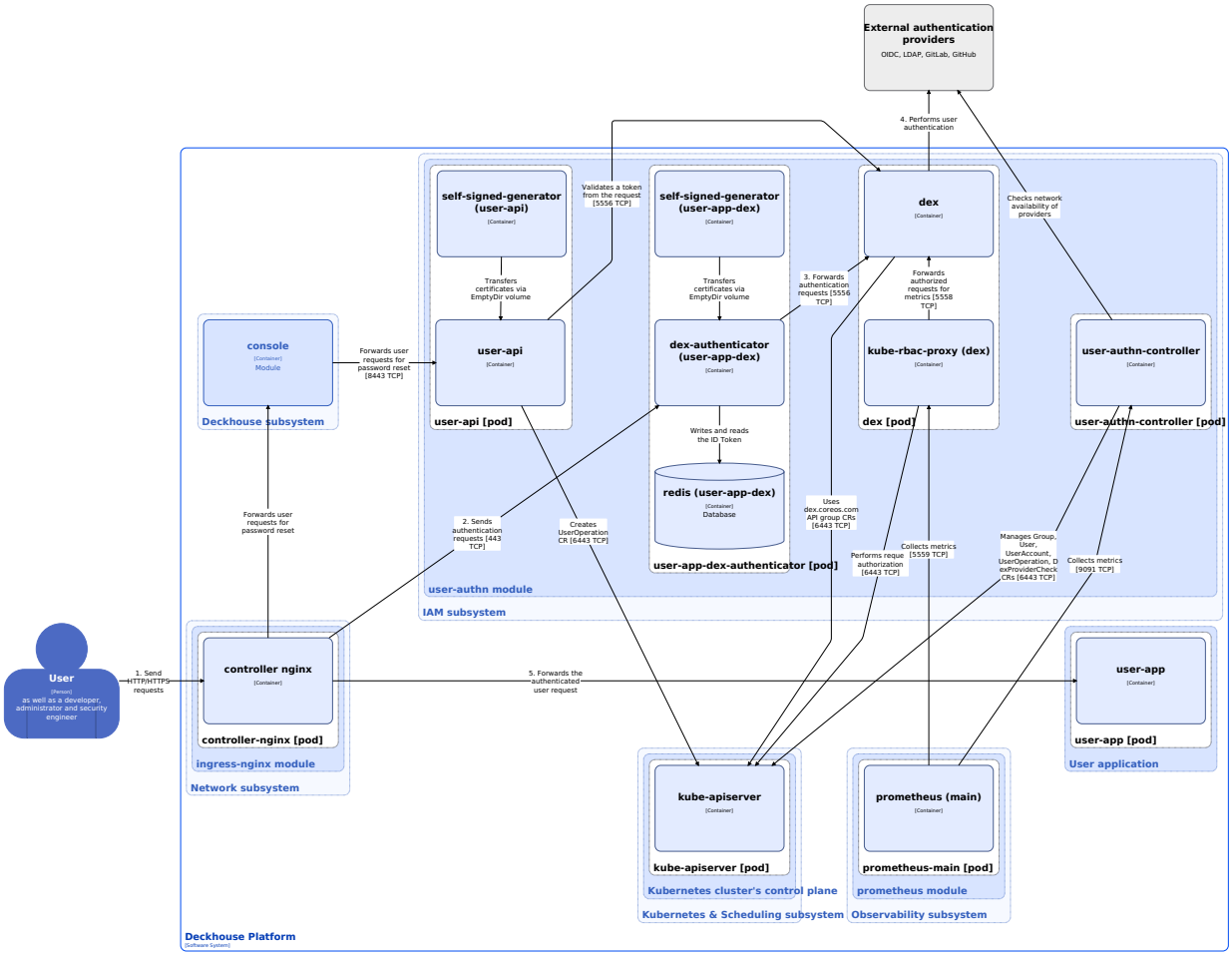
> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

In DP, two authentication schemes are used for platform services and user applications:

- Using dex-authenticator
- Using the Dex client

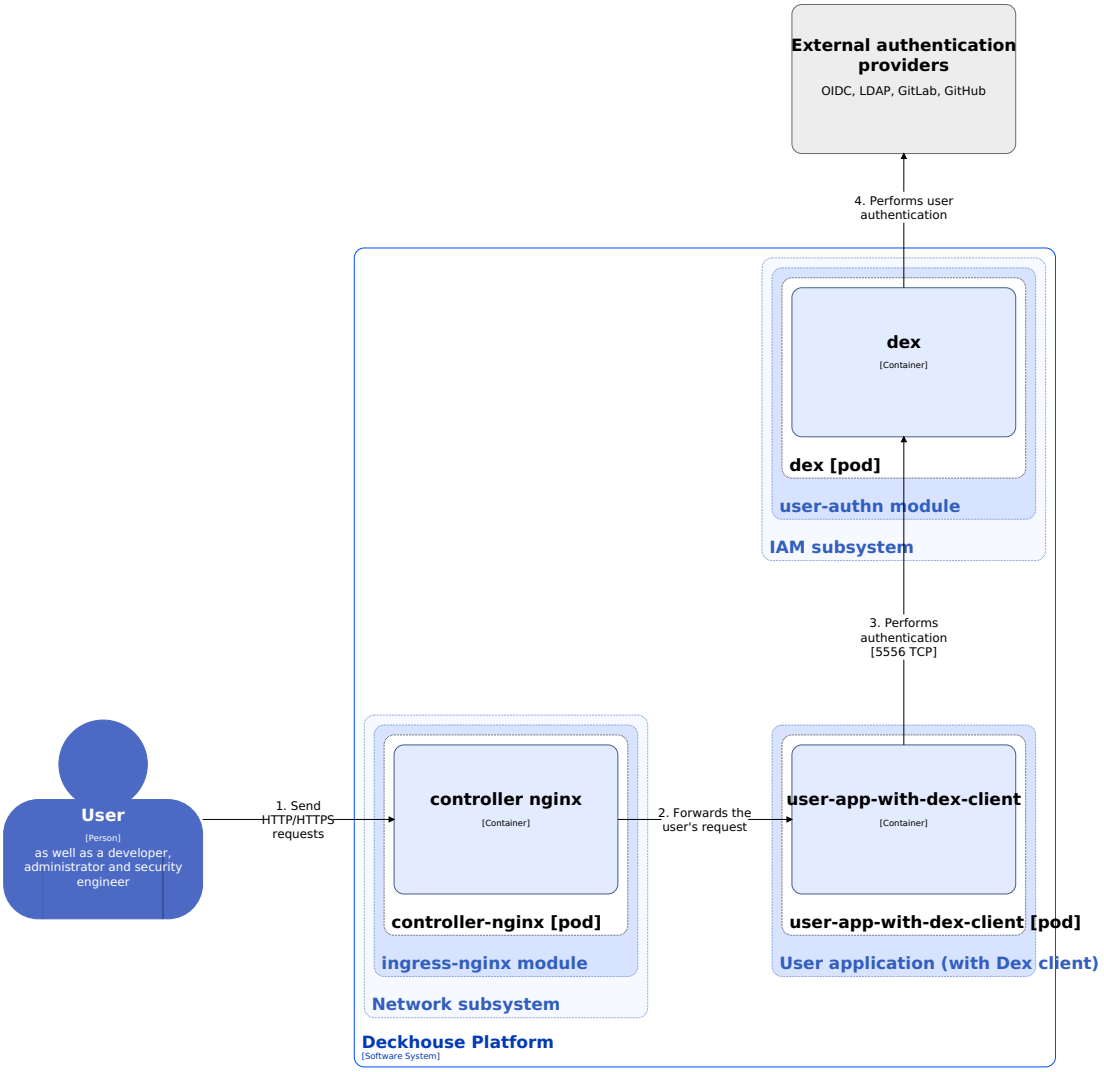
The architecture of the `user-authn` module at Level 2 of the C4 model and its interactions with other DP components are shown in the following diagrams.

Using dex-authenticator:



> **i** For simplicity, the following diagrams show only the components and their interactions that distinguish these diagrams from the variant using dex-authenticator.

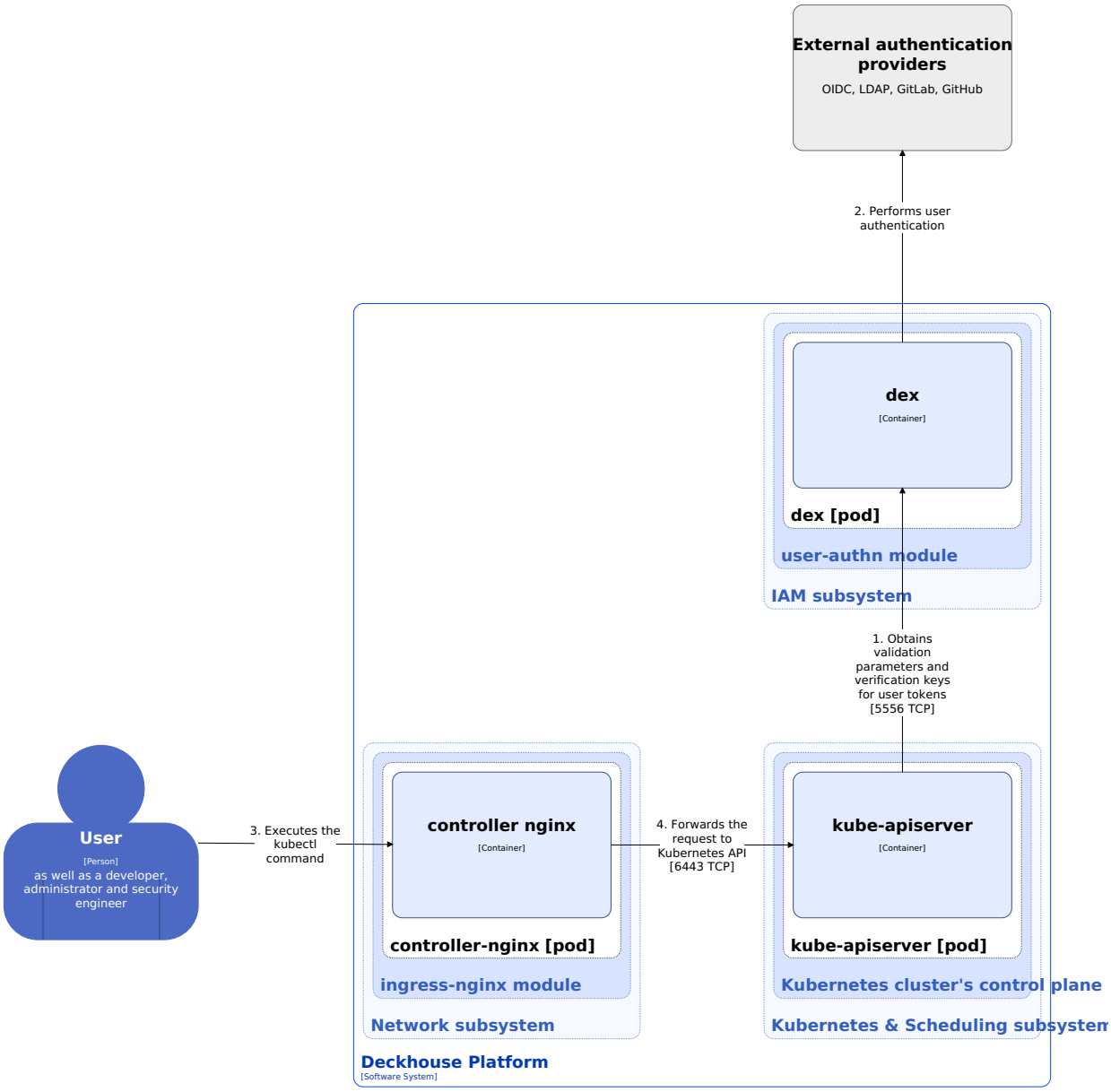
Using the Dex client:



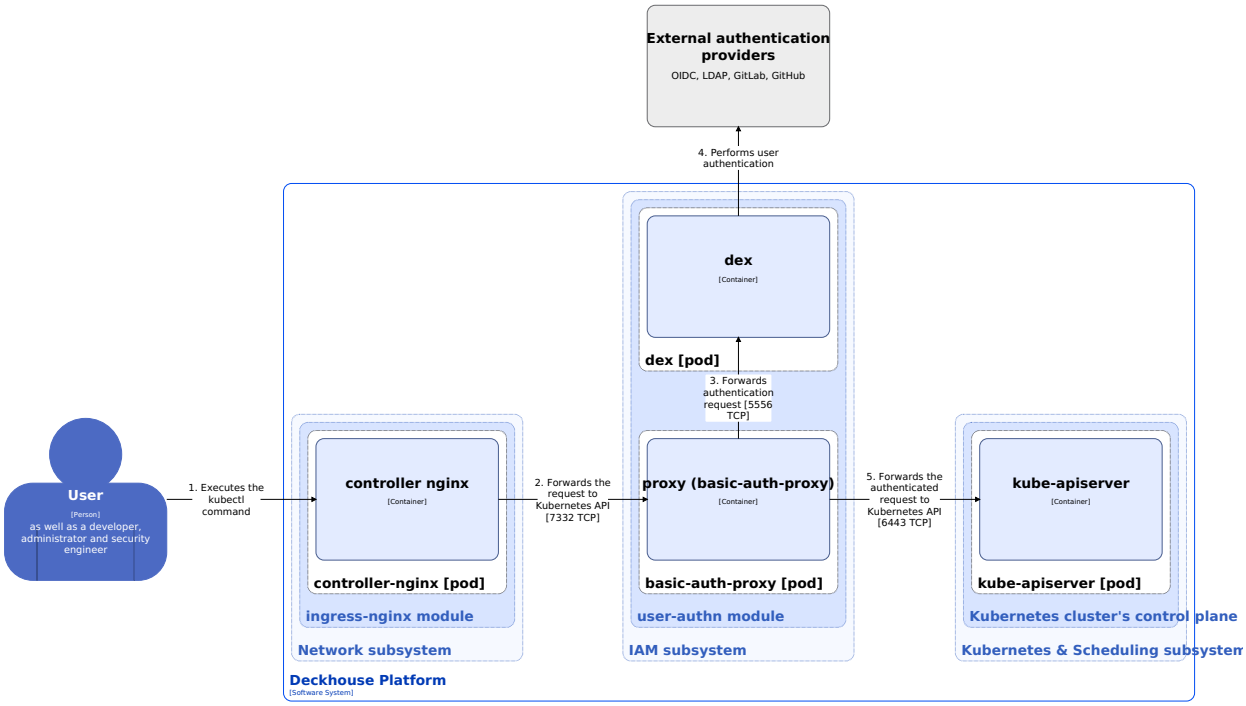
When connecting to the Kubernetes API using `d8`, `kubectl` or other Kubernetes clients with a generated kubeconfig, a separate authentication schemes are used:

- Token authentication. It is described in detail in the corresponding documentation section.
- Basic authentication. An example of configuring basic authentication using an LDAP provider is described in the module documentation section.

Using a generated kubeconfig and a token authentication:



Using a generated kubeconfig and a basic authentication:



Module components

The module consists of the following components:

1. **Dex**: Federated OpenID Connect provider that supports static users and integration with external authentication providers such as SAML, GitLab, or GitHub. The module uses a modified version of Dex to support:

   - Groups for static user accounts.
   - Two-factor authentication (2FA).
   - Password policies.
   - Forced password change.
   - Kerberos (SPNEGO) support for the LDAP connector. When enabled, Dex accepts `Authorization: Negotiate` tickets, validates them with a service keytab, and completes the login without rendering the password form, etc.

   For actual list of Dex modifications, refer to the module repository.

   It consists of the following containers:

   - **dex**: Main container implementing Dex functions.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to provider metrics. It is an open source project.

2. **Dex-authenticator**: Middleware service used to authenticate requests to applications through the DP cluster authentication service.

   When the Ingress controller is configured accordingly (using the NGINX `auth_request` module), requests are first forwarded to dex-authenticator for authentication.

   It consists of the following containers:

   - **dex-authenticator**: Main container of the service.
   - **redis**: Sidecar container with a Redis database used for temporary storage of ID tokens and fast access to them (since the database resides in memory).
   - **self-signed-generator**: Init container that generates a self-signed certificate when the pod starts.

3. **User-authn-controller**: A controller that consists of a single **user-authn-controller** container and performing following operations:

   - Manages module custom resources:
     - Group: A resource that describes user group.
     - User: A resource that describes static user.
     - UserAccount: A resource that describes view of Dex Password, OfflineSessions, and UserIdentity objects for the Deckhouse web UI.
     - UserOperation: A resource that describes an operation to be applied to a user (password reset, forced password change, 2FA reset, lock/unlock, logout).
     - DexProviderCheck: A resource that describes a connectivity check for the specific DexProvider. The check verifies that the provider exists, is enabled, Dex is reachable, and the provider endpoint is reachable. It does not perform a full user authentication flow.
   - Deletes expired Users (users whose `status.expireAt` field is in the past).
   - Checks periodically an availability of configured external authentication providers (using DexProviderCheck custom resources).
   - Finds out conflicting TLS-certificates in external LDAP provider configuration settings and exports the corresponding metric.
   - Revokes the sessions of a user on a lock, a password or 2FA reset, a logout and a user deletion: raises the generation of the user’s Dex account, writes a Dex Revocation object, and deletes the user’s Dex AuthSession and RefreshToken objects.
   - Serves the revocations to dex-authenticator over HTTPS (the `revocation-feed` Service), so that dex-authenticator ends the revoked sessions within seconds.
   - Keeps the `kubernetes` Dex client (OAuth2Client) and its `trustedPeers`: the DexAuthenticators and DexClients allowed access to Kubernetes, and the static clients of the module configuration.

   User-authn-controller uses `dex.coreos.com` API group custom resources (Password, OfflineSessions, UserIdentity, and RefreshToken used by Dex as a storage) as a backend to manage module custom resources.

4. **Dexclient-controller**: A controller that consists of a single **dexclient-controller** container (the image of user-authn-controller) and keeps the credentials of every DexClient: the `dex-client-<name>` Secret in the namespace of the DexClient and its Dex clients (OAuth2Client). It is the only component of the module that can write Secrets in other namespaces, and it serves nothing on the network but health and metrics.

5. **User-api**: A component that implements self-service operations for the user who is signed in to the DKP web interface: signing out on all devices and changing the password of a local account. It accepts only the ID tokens of the DKP web interface and does not require platform administrator rights. For

each operation, the component creates a UserOperation custom resource of the `Logout` or `RequirePasswordChange` type, which is processed by the user-authn-controller component. A ValidatingAdmissionPolicy allows user-api to create only these operations.

It consists of the following containers:

- **user-api**: Main container of the service.

6. **Basic-auth-proxy**: An optional component consisting of a single **proxy** container, which is launched when basic authentication is enabled in the settings of one of an external providers. When connecting to the Kubernetes API, the basic-auth-proxy component performs basic user authentication (through Dex for an LDAP provider, directly with the provider for a Crowd or OIDC provider), caches credentials validation results, and proxies authenticated requests to the Kubernetes API.

Module interactions

The module interacts with the following components:

1. **External authentication providers**.
2. **Kube-apiserver**:
   - Manages module custom resources.
   - Authorizes requests for metrics.

The following external components interact with the module:

1. **Ingress controller**: Forwards authentication requests to dex-authenticator for DP platform services and user applications.
2. **User applications**: Can authenticate directly with dex (without dex-authenticator) if an OAuth2 client is configured in Dex for the application. For more details about configuring a Dex client, refer to the `user-authn` module documentation.
3. **Kube-apiserver**: Queries dex when processing Kubernetes API requests made using a kubeconfig file:
   - kube-apiserver reads the OIDC provider configuration (Dex in this case) to obtain the `issuer` and the parameters required to validate tokens via the JWKS endpoint. In Kubernetes 1.30 and later, it reads it through the Dex Service inside the cluster.
   - When receiving a request with an ID token, kube-apiserver verifies its signature using the keys obtained from the JWKS endpoint, then it compares token claims against the server configuration.

   More details about connecting to the Kubernetes API using a generated kubeconfig can be found in the corresponding documentation page.
4. **Prometheus-main**: Collects metrics from the dex provider.
5. **Deckhouse web UI**: Forwards user requests for password reset.

Additional resources

- Module `console` documentation
- Module `user-authn` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

User-authz module

The `user-authz` module implements role-based access control (RBAC) in Deckhouse Platform (DP). The module creates cluster roles for managing user and group access to cluster resources, and, in the Ultimate, CSE Core, and CSE Pro editions of DP, also supports namespace-scoped authorization (multitenancy).

For more details about the role-based access control model, refer to the corresponding section of the module documentation.

The module works with the following custom resources of the `deckhouse.io` API group:

- ClusterAuthorizationRule: Defines access rules at the cluster level.
- AuthorizationRule: Defines access rules within a single namespace.

In the Ultimate, CSE Core, and CSE Pro editions of DP, the module additionally works with the following custom resources of the `authorization.deckhouse.io` API group:

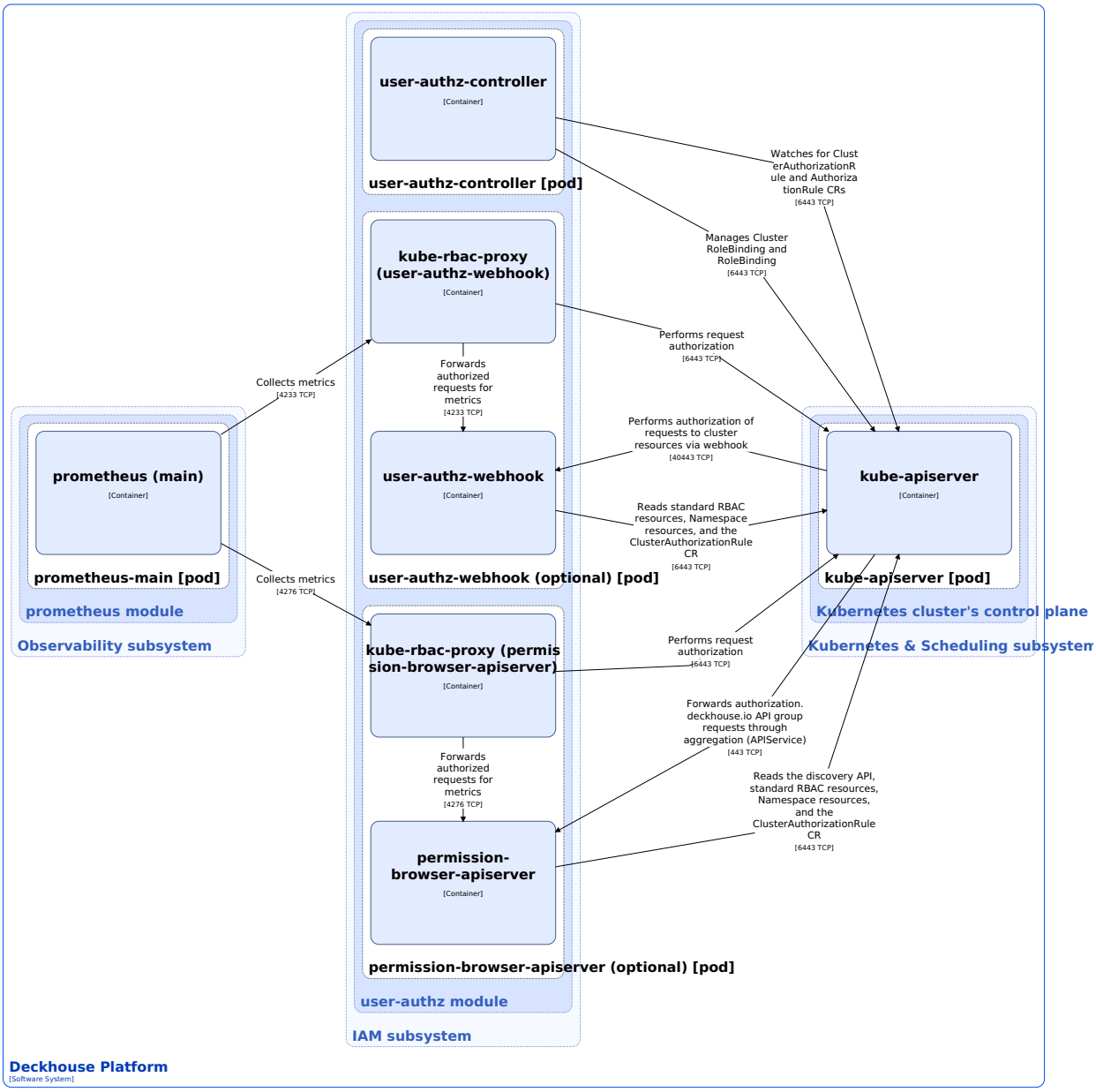
- BulkSubjectAccessReview: A resource for bulk-checking user permissions across multiple objects or actions.
- AccessibleNamespace: A resource for retrieving the list of namespaces accessible to a specific user.
- WhoCan: A resource for finding which users, groups, and service accounts are allowed to perform a given action.
- SubjectAccessReport: A resource for retrieving the full list of permissions granted to a subject.
- RoleAccessReport: A resource for retrieving the full list of resources and actions granted by a role.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following assumptions are made to simplify the diagram:
>
> - The diagram shows direct interaction between containers from different pods. In practice, they communicate through corresponding Kubernetes Services (internal load balancers). Service names are omitted when they are clear from context. In other cases, the service name is shown above the arrow.
> - Pods can run with multiple replicas, but all pods are shown as a single replica in the diagram.

The level 2 C4 architecture of the `user-authz` module and its interactions with other DP components are shown in the following diagram:



Module components

The module consists of the following components:

1. **User-authz-controller** (Deployment): This component includes a single **user-authz-controller** container and watches the ClusterAuthorizationRule and AuthorizationRule custom resources, creating, updating, and deleting the corresponding ClusterRoleBinding and RoleBinding resources.

The Ultimate, CSE Core, and CSE Pro editions of DP additionally include the following components:

1. **User-authz-webhook** (DaemonSet): An optional component that implements the Webhook authorization mode for kube-apiserver, placed between the built-in Node and RBAC authorizers in the authorization chain. The component runs on all master nodes with `hostNetwork` enabled.

   The webhook authorization mode for kube-apiserver is configured by the `control-plane-manager` module if the `.controlPlaneConfigurator.enabled` parameter in the module settings is set to `true` (the default). As part of this, the `control-plane-manager` module creates an AuthorizationConfiguration whose `matchConditions` parameter excludes the following subjects from being checked by the webhook:

   - Core control plane system identities, such as `kubernetes-admin`.
   - Node identities, `system:node:*`.
   - Service accounts from the `kube-system` and `d8-*` namespaces.

   Upon receiving a `SubjectAccessReview` request, the webhook checks the namespace access restrictions defined in the ClusterAuthorizationRule custom resource (multitenancy). It does not restrict a subject that has RBAC access regardless of the ClusterAuthorizationRule restrictions. This allows both access models to work together.

   The component can only deny a request explicitly; for every other request it returns no opinion, and the decision falls through to the RBAC authorizer. User-authz-webhook is fail-closed, so if the webhook is unavailable or does not respond in time, kube-apiserver denies every non-excluded request instead of falling back to RBAC.

   The following fail-closed parameters are used for the webhook:

   - `failurePolicy` is set to `Deny`.
   - Timeout: 3 seconds.
   - Decisions are cached separately for authorized and unauthorized requests (`authorizedTTL: 5m` / `unauthorizedTTL: 30s`).

   The Deckhouse controller deploys this component if the `.enableMultiTenancy` parameter in the module settings is set to `true` (default is `false`).

   Includes the following containers:

   - **user-authz-webhook**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to component metrics.

2. **Permission-browser-apiserver** (Deployment): An optional component, a Kubernetes API extension that publishes custom resources of the `authorization.deckhouse.io` API group.

   The BulkSubjectAccessReview, AccessibleNamespace, WhoCan, SubjectAccessReport, and RoleAccessReport custom resources of the `authorization.deckhouse.io` API group are used by the `console` module and provide extended capabilities for determining user, resource, or action permissions.

   The Deckhouse controller deploys this component if the `.enableMultiTenancy` parameter in the module settings is set to `true` (default is `false`).

   To expand wildcard RBAC rules into the cluster's actual API resources and determine whether each one is namespaced or cluster-scoped, it periodically queries the Kubernetes Discovery API endpoint and caches the result.

   Includes the following containers:

   - **permission-browser-apiserver**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to component metrics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches the ClusterAuthorizationRule and AuthorizationRule custom resources.
   - Manages the ClusterRoleBinding and RoleBinding resources.
   - Reads Namespace and RBAC resources, and queries the discovery API.
   - Authorizes requests for metrics.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Authorizes requests to cluster resources.
   - Forwards requests to the `authorization.deckhouse.io` API group.
2. **Prometheus-main**: Collects metrics from the user-authz-webhook and permission-browser-apiserver components.

Additional resources

- Module `console` documentation
- Module `control-plane-manager` documentation
- Module `user-authz` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Security subsystem

This subsection describes the architecture of the Security subsystem of Deckhouse Platform (DP).

The Security subsystem includes the following modules:

- `admission-policy-engine`: Lets you use security policies in the cluster according to Kubernetes Pod Security Standards. The module uses Gatekeeper to enforce these policies.
- `runtime-audit-engine`: Implements runtime audit (threat detection based on Linux kernel events and Kubernetes API audit events using Falco). The architecture is described in “Runtime audit”.
- `security-events-manager`: Performs declarative collection, processing, normalization, and delivery of security events from logs of applications and Kubernetes infrastructure components. The architecture is described in “Security events”.
- `operator-trivy`: Performs periodic vulnerability scanning of the DP cluster.
- `cert-manager`: Manages TLS certificates in the cluster.
- `secrets-store-integration`: Delivers secrets to Kubernetes applications by integrating secrets, keys, and certificates stored in external secret stores.
- `secret-copier`: Automatically copies secrets to cluster namespaces.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Integrity control

Integrity control is a set of mechanisms for verifying containers to ensure their security and compliance with the specified configuration.

Deckhouse Platform (DP) provides the following mechanisms:

- Integrity control of user workloads at startup: image signature verification during Kubernetes API request validation.
- Integrity control of running containers: runtime audit.
- Integrity protection of DP modules: modules are installed as immutable images.

Container image integrity control at the CRI level

Cryptographic image signature verification when a container image is pulled and a container starts, as well as protection of unpacked image layers from modification using DM-Verity, are implemented at the containerd level and are available **only in DP Certified Core and Certified Pro editions**. These editions use a modified version of containerd. The DM-Verity root hash is calculated for each image layer during the platform build and verified by containerd when the image is unpacked. Integrity control includes verification when image blocks are accessed and periodic re-verification:

- After unpacking, DM-Verity verifies image blocks whenever they are accessed, throughout the entire container runtime.

- In addition, containerd periodically (once per hour by default) rechecks running containers in protected namespaces. It verifies the image manifest signature again and fully checks image layers against the DM-Verity root hash (`veritysetup verify`). Layers are not checked all at once. The queue is processed at a rate of one container per minute, so a complete pass over a node takes longer as the number of containers on the node increases.

Platform image signatures are verified against a built-in set of public certificates in containerd. The signing private key belongs to the platform vendor, and you cannot add your own verification key. For more details about this mechanism, refer to the Certified Core and Certified Pro documentation.

In other DP editions (CE, BE, SE, SE+, EE, Ultimate, Core), image signature verification at the CRI level is not available:

- When using containerd v2, the EROFS snapshotter is used. Each OCI image layer is converted into a separate EROFS file and mounted read-only. This reduces the risk of tampering with data in images already unpacked on a node, but does not eliminate it completely and does not replace image authenticity verification. For details on switching to containerd v2, refer to “Migrating container runtime to containerd v2”.
- The SHA-256 hash verification performed by containerd when pulling an image protects against data corruption in transit, but does not confirm the image authenticity, because it does not make it possible to determine who built the image.

To control the integrity and authenticity of user application images in these editions, use signature verification during Kubernetes API request validation (for details, refer to “Integrity control of user workloads at startup”).

Integrity control of user workloads at startup

The integrity and authenticity of user application images are verified during Kubernetes API request validation, that is, before a Pod is created, rather than at the CRI level. The mechanism is implemented by the `admission-policy-engine` module and relies on signatures created with Cosign.

The sequence of integrity checks at startup:

1. At the build stage, the image is signed with a private key using Cosign. The signature is published in the container image registry as a separate tag and does not modify the image itself.
2. A SecurityPolicy resource with the `policies.verifyImageSignatures` parameter is created in the cluster, listing the public keys and the image address patterns.
3. When a Pod is created or updated, the validating webhook passes the request to Gatekeeper, which in turn queries the Ratify component.
4. Ratify verifies the image signature against the specified public keys.
5. If the signature is missing or does not match the specified keys, Pod creation is denied.

Additionally, you can restrict the list of container image registries that Pods are allowed to be started from using the `policies.allowedRepos` parameter of the OperationPolicy resource.

For more details on the configuration, refer to “Image signature verification”.

Integrity control of running containers

Runtime auditing in DP includes analyzing Linux kernel events and Kubernetes API audit events. This makes it possible to track whether applications in pods are running unchanged, conform to their expected state, and have not been modified.

Auditing uses:

- Built-in rules
- Custom rules that can be added using the Falco condition syntax

Integrity control of running containers can detect threats such as launching command-line shells inside containers or pods, discovering containers running in privileged mode, mounting insecure paths into containers, or attempts to read sensitive data.

For more details on configuring runtime audit, refer to “Runtime audit”.

Integrity protection of DP modules

Starting with version 1.74, DP modules are installed not as a directory with files, but as an image in the EROFS format that is mounted read-only through the DM-Verity kernel mechanism. This protects the contents of an already installed module from being modified on the node. The module files cannot be overwritten in the mounted image, and DM-Verity verifies the image blocks when they are accessed.

This is a separate mechanism unrelated to container image integrity control at the CRI level: here, the DP controller applies DM-Verity to the platform module image rather than containerd applying it to container image layers. Therefore, the edition restriction described in “Container image integrity control at the CRI level” does not apply to this mechanism.

The mechanism is enabled automatically if the `erofs` filesystem is registered in the kernel on the node

running the DP controller (a master node by default), that is, the corresponding kernel module is either loaded or built into the kernel. To check it, run the following command:

```
grep -w erofs /proc/filesystems
```

If the `erofs` filesystem is not available, DP will continue to operate and will install modules the regular way, without protecting their integrity. The switch happens without a dedicated alert, so make sure to check `erofs` availability in advance.

⚠ DP loads the `erofs` kernel module and configures it to load automatically only on nodes that use containerd v2 as the container runtime. If master nodes use containerd v1, the `erofs` module may end up not loaded, and DP modules will be installed without integrity protection, even if the kernel supports `erofs`.

To make sure the mechanism is engaged, either switch master nodes to containerd v2, or have the operating system load the `erofs` module.

Note the following when using this mechanism:

- The mechanism protects modules from being modified after installation, but does not confirm their authenticity: there is no cryptographic signature verification for modules. The container image registry the module is pulled from remains the source of trust.
- The mechanism applies only to DP modules and does not cover user application images.
- The mechanism is available in all DP editions and does not need to be enabled separately.

Additional resources

- Module `admission-policy-engine` custom resources

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Runtime audit architecture

The runtime audit of Deckhouse Platform (DP) is based on the Falco threat detection system. This mechanism analyzes Linux kernel events and Kubernetes API audit events to detect suspicious activity in running containers and across the cluster.

DP deploys Falco agents on each node as part of a DaemonSet. Once started, the agents begin collecting OS system calls and Kubernetes audit data.

i Falco developers recommend running it as a systemd service, which can be challenging in Kubernetes clusters that support autoscaling. DP includes additional security mechanisms such as multitenancy and resource control policies. Combined with the DaemonSet deployment, these mechanisms ensure a high level of protection.

Each cluster node runs a Falco Pod with the following components:

- `falco`: Collects events, enriches them with metadata, and outputs them to stdout.
- `rules-loader`: Retrieves rule data from FalcoAuditRules custom resources and stores them in a shared directory.
- `falcosidekick`: Receives events from `falco` and exports them as metrics to external systems.
- `kube-rbac-proxy`: Protects the `falcosidekick` metrics endpoint from unauthorized access.

For a detailed architecture of the `runtime-audit-engine` module, which implements DP security event audit, refer to the corresponding documentation section.

Audit rules

Runtime audit event analysis is performed using rules that define suspicious behavior patterns. Each rule consists of a condition expression written in accordance with Falco's condition syntax.

Built-in rules

DP provides the following types of built-in rules:

- **Kubernetes audit rules**: Help detect security issues in DP and in the audit mechanism itself. These rules are located in the `falco` container at `/etc/falco/k8s_audit_rules.yaml`.

Custom rules

Custom rules can be defined using the FalcoAuditRules custom resource.

Each Falco agent includes a sidecar container with a `shell-operator` instance. This instance reads rules from Kubernetes resources, converts them into Falco rule format, and stores them in the `/etc/falco/rules.d/` directory inside the Pod. When a new rule is added, Falco automatically reloads the configuration.

Additional resources

- Module `runtime-audit-engine` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Security events architecture

The `security-events-manager` module within Deckhouse Platform (DP) performs declarative collection, processing, normalization, and delivery of security events extracted from logs of applications and Kubernetes infrastructure components.

A security event is a structured record of an action or fact that is significant from an information security perspective. Typical categories of such events include:

- Authentication and authorization
- Access to APIs and configuration
- Changes to cluster objects
- Runtime and network activity anomalies

Regardless of the original log format, the output is a uniform event model with a mandatory minimum set of attributes.

> ⚠ The `security-events-manager` module is in the `Experimental` stage and may change in future releases.

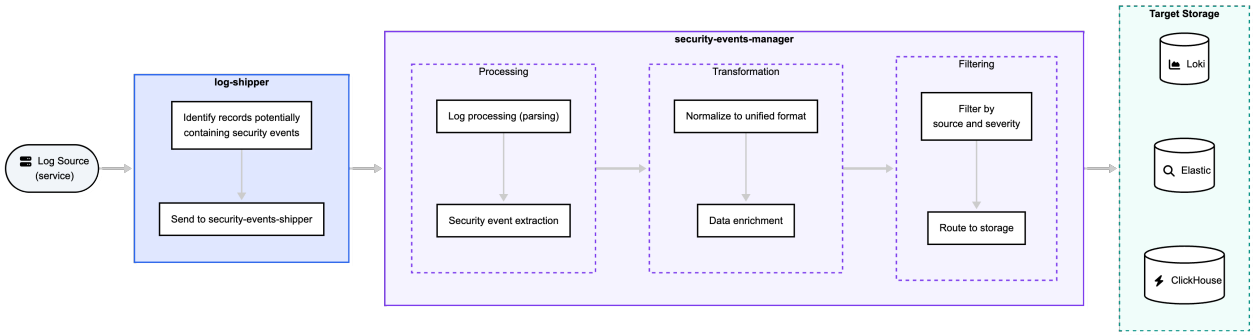
Solution architecture

The architecture is intended to build a unified pipeline for working with security events extracted from the logs of applications and Kubernetes infrastructure components. The module performs three processing stages:

1. **Collection**: Collecting logs from pod sources and node files, initial selection of records that may contain security events.
2. **Processing and enrichment**: Parsing logs, extracting event signals, transforming into a unified model, and enriching with context.
3. **Delivery**: Policy-based filtering and delivery to configured storages.

At the delivery stage, sending to several types of storage and analytics systems is supported (such as Loki, Elasticsearch, Kafka, Splunk, Vector, File).

Processing pipeline



The architecture separates three stages of the pipeline: **collection**, **processing and enrichment**, and **delivery**.

Collection layer (log-shipper)

Collection is performed via the auxiliary `log-shipper` module:

- Container sources use application logs within the namespace.
- Cluster sources use node files and system service logs (for example, `/var/log/kube-audit/audit.log`, `/var/log/auth.log`).

A two-tier scheme is used:

- The `log-shipper` module performs preliminary selection of log records using simple comparison operators (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) and forwards them to the gateway.
- The `security-events-manager` module (gateway) performs field recognition (parsing), further processing, and delivery.

Because log parsing is resource-intensive, it is performed only for records pre-selected as potentially containing security events, not for all incoming logs. Therefore, the initial selection stage does not perform deep field-value filtering that requires full content parsing.

Example of an input log for the selection stage:

```
{
  "time": "2026-05-10T14:21:03Z",
  "kind": "Event",
  "source": "kube-apiserver",
  "level": "Metadata",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401,
  "requestURI": "/api/v1/namespaces/default/secrets",
  "user": "system:serviceaccount:default:demo",
  "sourceIPs": [
    "206.123.145.70"
  ]
}
```

Processing and extraction

After being forwarded to the gateway, the log structure is recognized. Standard processing strategies are supported:

- `JSON`: For structured logs.
- `Regex`: For string formats with predictable patterns.
- `Grok`: For complex, non-unified formats.

The processing result is used to derive event signals and build a unified set of fields.

Example output of the processing stage:

```
{
  "parsed": {
    "timestamp": "2026-05-10T14:21:03Z",
    "source_component": "kube-apiserver",
    "http_status": 401,
    "request_uri": "/api/v1/namespaces/default/secrets",
    "actor_id": "system:serviceaccount:default:demo",
    "source_ip": "206.123.145.70"
  }
}
```

After processing, the record becomes a source of fields for event classification (code/category/severity/outcome), as well as for building context (`actor`, network attributes, and source metadata).

Transformation and enrichment

Transformation is implemented in two steps:

1. **Transform**: Mapping fields from the original log to the fields of the target event model.
2. **Enrich**: Adding or refining fields from additional context sources (for example, static environment attributes, subject roles, and service indicators).

The order is fixed: `Transform` is applied first, then `Enrich`. In case of a conflict for a target field, the final value is determined by the `Enrich` stage.

Filtering and delivery

After the event is formed, a delivery policy is applied:

- Source filtering
- Filtering by minimum severity
- Routing to one or more destinations

Filtering rules can use both exact source identifiers and source masks, which makes it possible to manage flows at the level of individual services or entire groups.

Destinations can be storage and processing systems such as cluster Loki, external SIEM/log platforms, and streaming buses. The delivery scheme supports parallel delivery to multiple target systems.

Security event model

Below is the structure of the event that is sent to storage.

Required fields:

- `id`, `timestamp`
- `source.component`
- `event.code`, `event.category`, `event.severity`, `event.outcome`
- `eventMetadata.cluster`

Optional fields:

- `eventMetadata.sourceIPs`
- `actor.*`
- `object.*`

```
{
  "id": "2f0de5c2-2e58-4d3f-b4fe-5ec6f1935b9f",
  "timestamp": "2026-05-10T14:21:03Z",
  "source": {
    "component": "kube-apiserver"
  },
  "event": {
    "code": "UNAUTHORIZED_ACCESS",
    "category": "Rbac",
    "severity": "High",
    "outcome": "Failure"
  },
  "eventMetadata": {
    "cluster": "prod-cluster",
    "sourceIPs": [
      "206.123.145.70"
    ]
  },
  "actor": {
    "id": "system:serviceaccount:default:demo",
    "type": "ServiceAccount"
  },
  "object": {
    "id": "/api/v1/namespaces/default/secrets",
    "type": "KubernetesResource"
  }
}
```


Minimal sufficient lifecycle

In a practical scenario, the architecture works along the following chain:

1. Log sources are connected to the collection pipeline.
2. An initial selection of potentially relevant records is performed.
3. Records are processed and transformed into unified events.
4. Events are enriched with contextual attributes.
5. Filtering and routing rules are applied.
6. Events are delivered to target storages and analytics systems.

The result is a unified and manageable flow of security events suitable for monitoring, investigations, and long-term auditing.

Supported security events

The `security-events-manager` module ships with a built-in set of security event detection rules covering authentication, configuration, RBAC, runtime, and other categories. For the up-to-date list of implemented security events, their codes, severity levels, and descriptions, refer to the module documentation.

Additional resources

- Module `log-shipper` documentation
- Module `security-events-manager` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Admission-policy-engine module

The `admission-policy-engine` module enforces security policies and operational restrictions in a Kubernetes cluster. Policies are applied based on Pod Security Standards checks and rules from the following custom resources:

- OperationPolicy: Describes the operational policy of the cluster.

- SecurityPolicy: Describes the security policy of the cluster.
- SecurityPolicyException: Describes exceptions to the cluster security policy.

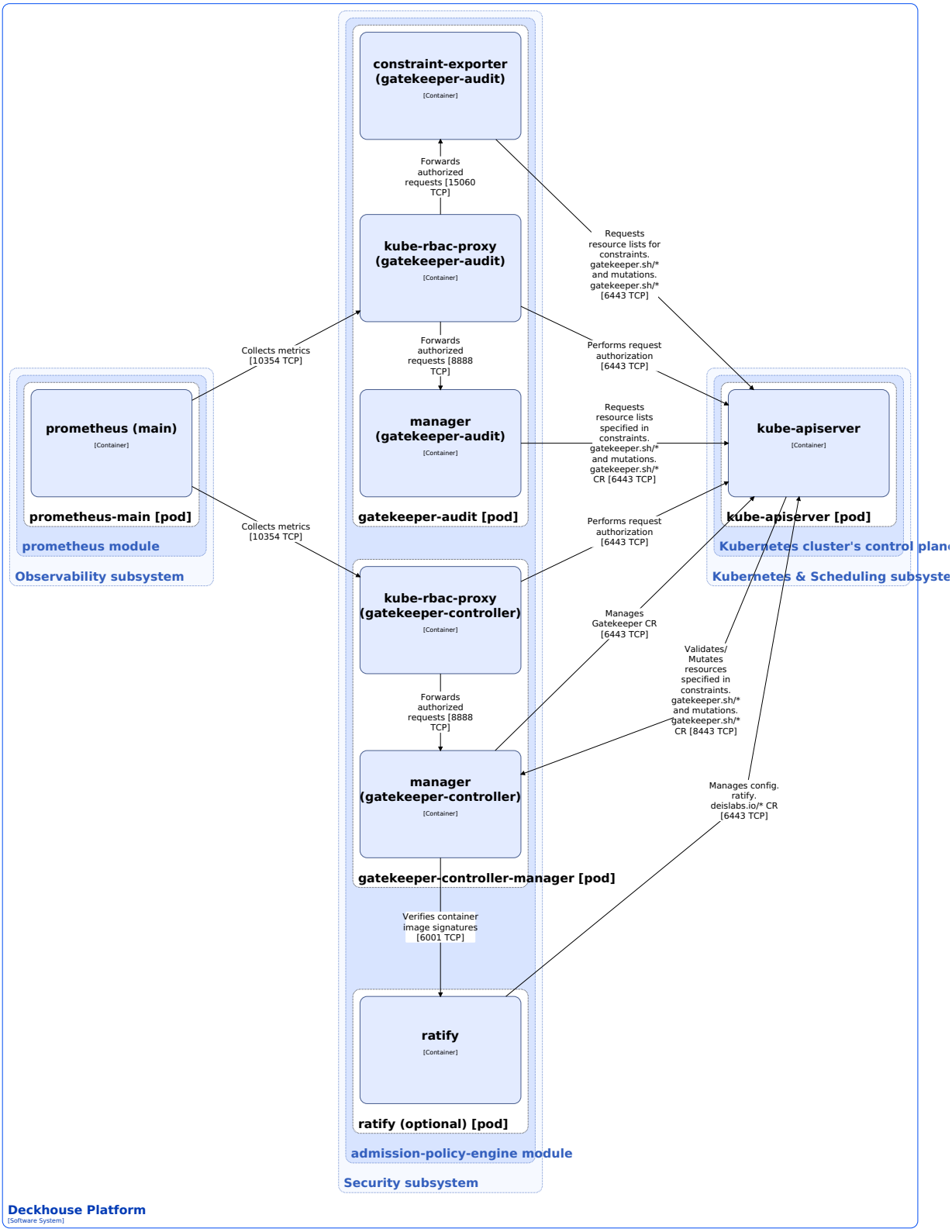
> **i** The `deckhouse` module processes the OperationPolicy and SecurityPolicy resources. The Deckhouse controller of the `deckhouse` module uses addon-operator and module hooks to create custom resources for Gatekeeper based on OperationPolicy and SecurityPolicy. Gatekeeper uses the resulting custom resources to validate newly created or updated Kubernetes resources.
>
> For details on the hooks concept, refer to the addon-operator documentation.

For a detailed description of the module, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `admission-policy-engine` module and its interaction with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Gatekeeper-controller-manager**: A Gatekeeper controller that performs the following operations:

   - Manages Gatekeeper custom resources.
   - Validates Kubernetes resources specified in custom resources from the `constraints.gatekeeper.sh/*` API group.
   - Mutates Kubernetes resources specified in the AssignMetadata, Assign, ModifySet, and AssignImage custom resources.

   Security rules are defined using the ConstraintTemplate custom resource and custom resources from the `constraints.gatekeeper.sh/*` API group. A ConstraintTemplate defines new policy types, based on which specific security policies are created to validate resources.

It consists of the following containers:

- **manager**: Main container.
- **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

2. **Gatekeeper-audit**: Implements periodic checks of existing Kubernetes resources for compliance with security policies.

   It consists of the following containers:

   - **manager**: Main container.
   - **constraint-exporter**: Sidecar container that exposes additional metrics for the `constraints.gatekeeper.sh/*` and `mutations.gatekeeper.sh/*` custom resources.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to metrics from `manager` and `constraint-exporter`.

3. **Ratify**: An optional component consisting of a single **ratify** container. It provides a Gatekeeper provider implementation for validating metadata of used artifacts. In DP, this provider is used to verify container image signatures and is available in the SE+, EE, CSE Lite, and CSE Pro editions.

> **i** Gatekeeper uses the Provider custom resource to extend resource validation capabilities in Kubernetes. The Provider resource describes the service endpoint to which Gatekeeper sends requests during ValidationWebhook execution. Some DP modules, such as `operator-trivy`, can create Provider custom resources and thereby extend the verification capabilities.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Monitors Kubernetes resources specified in custom resources of the `constraints.gatekeeper.sh/*` and `mutations.gatekeeper.sh/*` API groups.
  - Manages ConstraintTemplate, Assign, AssignImage, AssignMetadata, ModifySet, as well as resources from the `constraints.gatekeeper.sh/*` and `config.ratify.deislabs.io/*` API groups.

The following external components interact with the module:

1. **Kube-apiserver**: Validates Kubernetes resources and checks their compliance with the defined security rules.
2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `admission-policy-engine` documentation
- Module `deckhouse` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

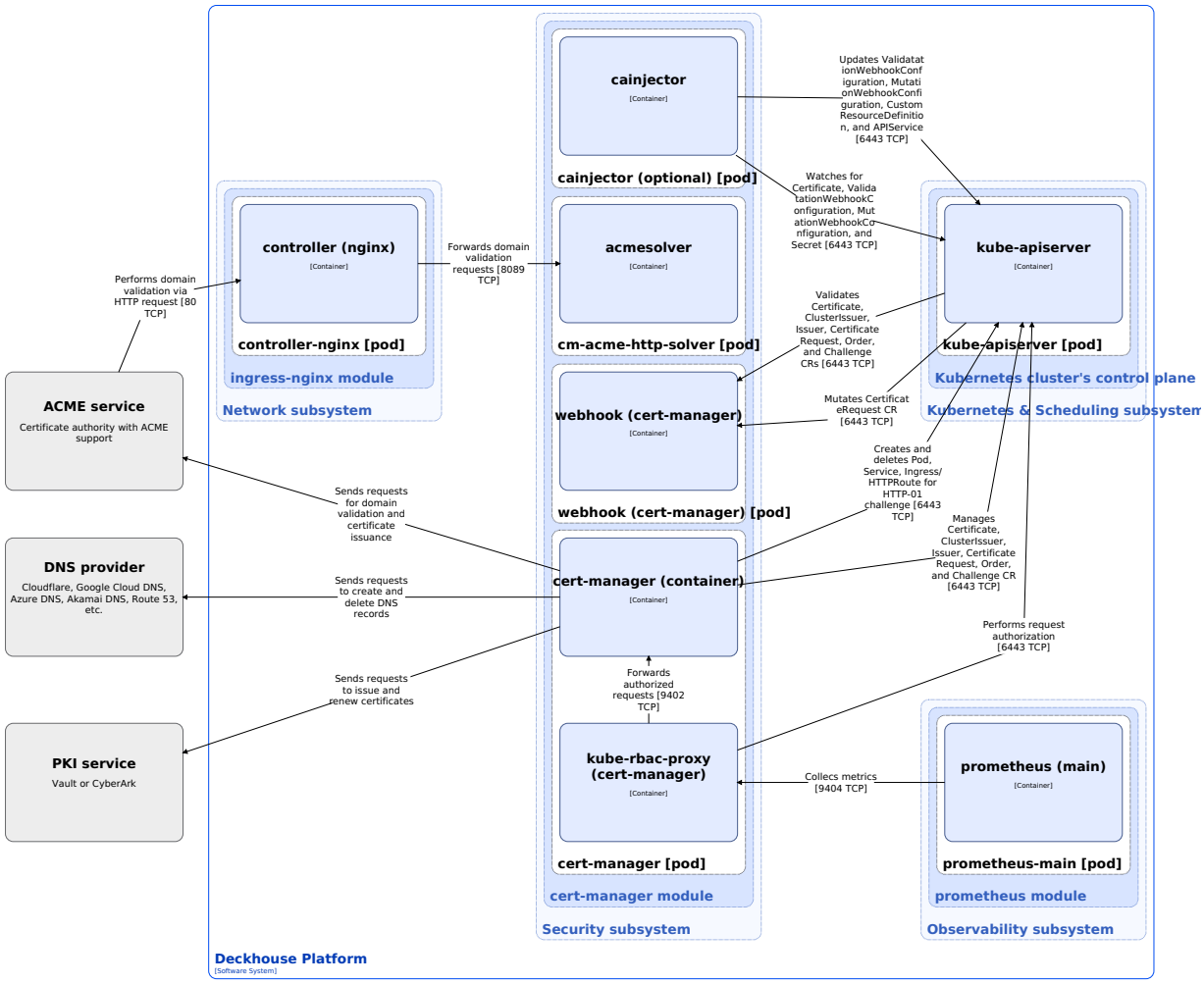
Cert-manager module

The `cert-manager` module automates the full certificate management lifecycle in a cluster: from issuing and renewing self-signed certificates to integration with external certificate authorities such as Let’s Encrypt, HashiCorp Vault, and Venafi. This significantly simplifies service security and provides centralized control over all certificate-related processes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cert-manager` module and its interactions with other DP components are shown in the following diagram:



Module components

The `cert-manager` module consists of the following components:

1. **Cert-manager**: A controller that provides the full certificate management lifecycle in Deckhouse Platform (DP). `Cert-manager` manages the following custom resources:
   - Issuer: Describes settings and parameters for obtaining certificates from a specific source, such as a CA or an external service. It is used within a selected namespace.
   - ClusterIssuer: Similar to Issuer, but applies to the entire cluster and is available in all namespaces.
   - Certificate: Defines the required certificate and specifies parameters such as the subject, validity period, the Issuer or ClusterIssuer used, and additional options.
   - CertificateRequest: A request to issue or renew a certificate.
   - Challenge: Describes a task used to complete domain validation, such as an HTTP-01 or DNS-01 challenge for Let’s Encrypt.
   - Order: Groups related Challenges into a sequence for obtaining a certificate from an ACME server, such as Let’s Encrypt.

   The component includes the following containers:
   - **cert-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the cert-manager container metrics.

> **i** To complete the DNS-01 challenge, `cert-manager` supports a number of popular DNS providers, such as AzureDNS, Cloudflare, DigitalOcean, and others. For the full list of supported DNS providers, refer to the cert-manager documentation. For providers that are not supported out of the box, webhook issuer types are used as external components. To work correctly, such providers must be installed in the `d8-cert-manager` system namespace. When making changes to this namespace, it is recommended to always account for such extensions so they remain operational.

2. **Webhook**: A component consisting of a single webhook container and performing the following operations:
   - validates the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources
   - mutates CertificateRequest custom resources by adding the user identity created the certificate request

   In DP, validation is disabled for resources in the `d8-cert-manager` namespace and for namespaces with the `cert-manager.io/disable-validation=true` label.
3. **Cainjector**: An additional component consisting of a single cainjector container. Cainjector automatically injects or updates root certificate authority (CA) certificates in all relevant Kubernetes resources: ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition, and APIService. This keeps trusted root certificates up to date for services that use webhooks and API extensions.

   Cainjector is enabled using the `.spec.settings.enableCAInjector` parameter in the `cert-manager` module settings. DP does not use cainjector, so enable it only if your services use custom CA injections.

   Cainjector processes only resources with the `cert-manager.io/inject-ca-from`, `cert-manager.io/inject-ca-from-secret`, or `cert-manager.io/inject-apiserver-ca` annotations, depending on the resource type.
4. **Cm-acme-http-solver**: A temporary pod with the acmesolver container that is launched to complete an HTTP-01 Challenge during domain validation through ACME, such as Let’s Encrypt. This pod is automatically created by `cert-manager` controller for the duration of the HTTP-01 Challenge and is deleted after the procedure is complete. This approach provides secure temporary publication of the resource that confirms domain ownership for certificate issuance.

Module interactions

The `cert-manager` module interacts with the following components:

1. **Kube-apiserver**:
   - Manages the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources.
   - Watches and updates ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition, and APIService resources.
2. **ACME service**: Handles requests for domain verification and certificate issuance.
3. **PKI service**: Handles requests for issuing and renewing certificates.
4. **DNS provider**: Handles requests to add and delete records in DNS services to complete DNS-01 Challenge during domain validation through ACME server.

The following external components interact with the module:

1. **Kube-apiserver**:
   - validates the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources;
   - mutates CertificateRequest custom resources.
2. **Prometheus-main**: Collects `cert-manager` metrics.
3. **Nginx Controller**: Forwards requests from ACME server to `cm-acme-http-solver`.

Additional resources

- Module `cert-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module operator-trivy

The `operator-trivy` module scans user images at runtime for known CVEs (Common Vulnerabilities and Exposures), including vulnerabilities specific to Astra Linux, ALT Linux, and RED OS. It is based on the Trivy project. Scanning uses public vulnerability databases, enriched with Astra Linux, ALT Linux, RED OS databases, and BDU FSTEC (Data Bank of Information Security Threats by the Federal Service for Technical and Export Control of Russia).

The module also analyzes Kubernetes cluster compliance with the CIS (Center for Internet Security) Kubernetes Benchmark requirements.

Additionally, the `operator-trivy` module can collect workload behavior models when the optional node-agent component is enabled.

The `operator-trivy` module works with custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.

The `trivy.deckhouse.io` API group includes the following resources:

- ClusterComplianceReport: Stores an aggregated cluster compliance report for security requirements (for example, CIS Kubernetes Benchmark).
- ClusterConfigAuditReport: Stores cluster-level audit results for Kubernetes object configuration.
- ClusterInfraAssessmentReport: Stores cluster-level Kubernetes infrastructure security assessment results.
- ClusterRbacAssessmentReport: Stores cluster-level role-based access control (RBAC) settings assessment results.
- ClusterSbomReport: Stores an aggregated cluster-level SBOM (Software Bill of Materials) for software components.
- ClusterVulnerabilityReport: Stores aggregated cluster-level vulnerability scanning results.
- ConfigAuditReport: Stores audit results for Kubernetes object configuration.
- ExposedSecretReport: Stores results of searching for potential secrets in container images.
- InfraAssessmentReport: Stores security assessment results for Kubernetes infrastructure objects.
- NodeVulnerabilityReport: Stores vulnerability scanning results for node `rootfs` (the root filesystem).
- RbacAssessmentReport: Stores RBAC settings assessment results for excessive privileges and other risks.
- RegistryImageVulnerabilityReport: Stores CVE scanning results for a specific image tag in a registry.
- RegistryScanTarget: Defines a target registry, repositories, and periodic scanning parameters.
- SbomReport: Stores an SBOM, that is, the software and dependency composition of a container image.
- VulnerabilityReport: Stores vulnerability scanning results for container images.

The `spdx.softwarecomposition.kubescape.io` API group includes the following resources:

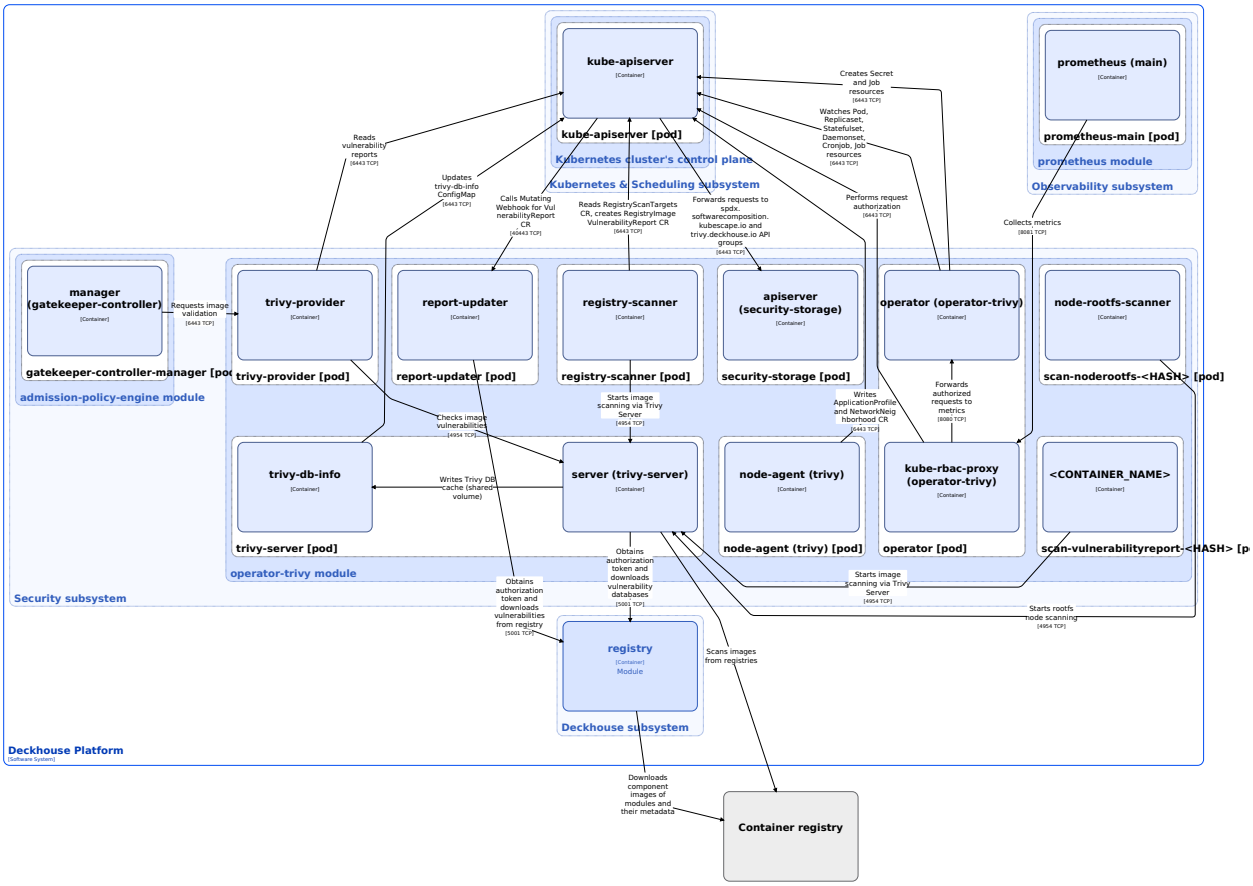
- ApplicationProfiles: Stores an application behavior profile at runtime (system calls, executed processes, file access, HTTP endpoints).
- CollapseConfigurations: Defines dynamic path, endpoint, and network address aggregation parameters to reduce runtime profile size.
- ConfigurationScanSummaries: Stores a summary of configuration scan results for a workload group within a specified scope (for example, a namespace).
- ContainerProfiles: Stores an individual container profile, including runtime behavior and network interactions.
- GeneratedNetworkPolicies: Stores NetworkPolicy objects generated from observed traffic.
- KnownServers: Stores a catalog of known servers and IP address ranges to enrich generated network policies.
- NetworkNeighborhoods: Stores an observed map of incoming and outgoing workload network interactions.
- OpenVulnerabilityExchangeContainers: Stores OpenVEX documents with vulnerability statuses for components.
- SbomSyftFiltereds: Stores a filtered Syft SBOM with only relevant vulnerable components.
- SbomSyfts: Stores SBOM in Syft format.
- SeccompProfiles: Stores container seccomp profiles (allowed system calls and filtering rules).
- VulnerabilityManifestSummaries: Stores VulnerabilityManifest summaries by severity level with links to related objects.
- VulnerabilityManifests: Stores a detailed manifest of discovered vulnerabilities.
- VulnerabilitySummaries: Stores an aggregated vulnerability summary for a specified scope.
- WorkloadConfigurationScans: Stores detailed configuration scan results for a specific workload.
- WorkloadConfigurationScanSummaries: Stores a summary of WorkloadConfigurationScan results.

For more details, see the module documentation section.

Module architecture

> The following assumptions are made to simplify the diagram:
>
> - The diagram shows direct interaction between containers from different pods. In practice, they communicate through corresponding Kubernetes Services (internal load balancers). Service names are omitted when they are clear from context. In other cases, the service name is shown above the arrow.
> - Pods can run with multiple replicas, but all pods are shown as a single replica in the diagram.

The level 2 C4 architecture of the `operator-trivy` module and its interactions with other Deckhouse Platform (DP) components are shown in the following diagram:



Module components

The module consists of the following components:

1. **Operator** (Deployment): A component based on Trivy Operator that watches Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources, and starts a Job task for scanning when their specifications change.

   Operator analyzes Job execution results and generates the following reports as custom resources:

   - ClusterComplianceReport.
   - ClusterConfigAuditReport.
   - ClusterInfraAssessmentReport.
   - ClusterRbacAssessmentReport.
   - ClusterVulnerabilityReport.
   - ConfigAuditReport.
   - ExposedSecretReport.
   - InfraAssessmentReport.
   - NodeVulnerabilityReport.
   - RbacAssessmentReport.
   - SbomReport.
   - VulnerabilityReport.

   Includes the following containers:

   - **operator**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to controller metrics.

2. **Trivy-server** (StatefulSet): This component implements a security scanning service and is based on the Open Source Trivy project.

   Trivy-server updates its vulnerability database from an Open Container Initiative (OCI) image at startup and then periodically.

   Trivy-server processes requests from other module components (operator, registry-scanner, trivy-provider, and scanning tasks), performs the requested scan, and returns the result.

   Includes the following containers:

   - **server**: Main container.
   - **trivy-db-info**: Sidecar container that synchronizes Trivy cache metadata with the `trivy-db-info` ConfigMap resource in the `d8-operator-trivy` namespace.

3. **Trivy-provider** (StatefulSet): This component includes a single **trivy-provider** container and provides an image verification interface for the Gatekeeper component of the `admission-policy-engine` module.

   When the module is installed, a Provider resource is created to register trivy-provider as a provider in Gatekeeper. For details about the integration, see the relevant Gatekeeper documentation section.

The Deckhouse controller deploys this component if the `.settings.denyVulnerableImages.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`) and the Gatekeeper component of the `admission-policy-engine` module is enabled.

4. **Report-updater** (Deployment): An optional controller that includes a single **report-updater** container, implements a MutatingWebhook, and enriches the VulnerabilityReport custom resource with BDU identifiers. The BDU dictionary is regularly updated from an OCI image every six hours.

   The Deckhouse controller deploys this component if the `.settings.linkCVEtoBDU` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

5. **Registry-scanner** (StatefulSet): An optional component that includes a single **registry-scanner** container and regularly scans images located in arbitrary container registries specified by the user. Scanning of workloads in the cluster itself is performed by the operator component.

   The Deckhouse controller deploys this component if the `.settings.registryScanning.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

   Registry-scanner reads RegistryScanTarget custom resources, runs image scans through the trivy-server component, and stores processed results in the RegistryImageVulnerabilityReport custom resource.

6. **Security-storage** (Deployment): A controller that includes a single **apiserver** container, extends Kubernetes API, and processes requests for custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.

   Security-storage also implements the backend for storing these resources:

   - Metadata is stored in an SQLite database.
   - Object body is stored in gob format in the `/data` directory.

7. **Node-agent** (DaemonSet): An optional component with a single **node-agent** container that runs on all cluster nodes in privileged mode, observes container behavior by using eBPF programs, and builds runtime profiles.

   Node-agent stores runtime profiles in ApplicationProfile and NetworkNeighborhood custom resources (in the security-storage component). For details about node-agent behavior, see the module documentation section.

   > ⚠ The node-agent component has privileged access to the operating system of each node. In Linux, this requires the following capabilities:
   >
   > - SYS_ADMIN
   > - SYS_PTRACE
   > - NET_ADMIN
   > - SYSLOG
   > - SYS_RESOURCE
   > - IPC_LOCK
   > - NET_RAW
   >
   > This is required to observe container behavior by using eBPF programs.
   >
   > Node-agent runs in profiling mode and does not perform active rule-based attack detection.

   The Deckhouse controller deploys this component if the `.settings.nodeAgent.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

8. **Scan-noderootfs-<HASH>** (Job): A component that includes a single **node-rootfs-scanner** container and implements a task to scan the node root filesystem (scan-noderootfs). The task is created and managed by the operator component.

9. **Scan-vulnerabilityreport-<HASH>** (Job): A component that launches tasks for security scanning of Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources by using the trivy-server component. The task is created and managed by the `operator` component.

   Consists of a set of **<CONTAINER_NAME>** containers, each responsible for scanning the corresponding workload container. Their base image is Trivy with an added trivy-wrapper. The target container image from the workload specification is passed in arguments. Trivy-wrapper authenticates to the registry storage by using the `trivy registry login` command and then hands control over to Trivy.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests to operator metrics.
   - Works with custom resources from the `trivy.deckhouse.io` API group.
   - Works with RegistryScanTarget and RegistryImageVulnerabilityReport custom resources.
   - Watches Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources.
   - Creates, updates, and deletes Secret, ConfigMap, and Job resources.
2. **Registry**: Pulls OCI images with the vulnerability database and the BDU database.
3. **Container registry**: Scanning images.

The following external components interact with the module:

1. **Kube-apiserver**:

    - Forwards API requests for custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.
    - Mutates VulnerabilityReport custom resources.

2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `admission-policy-engine` documentation
- Module `operator-trivy` documentation
- Module `registry` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Runtime-audit-engine module

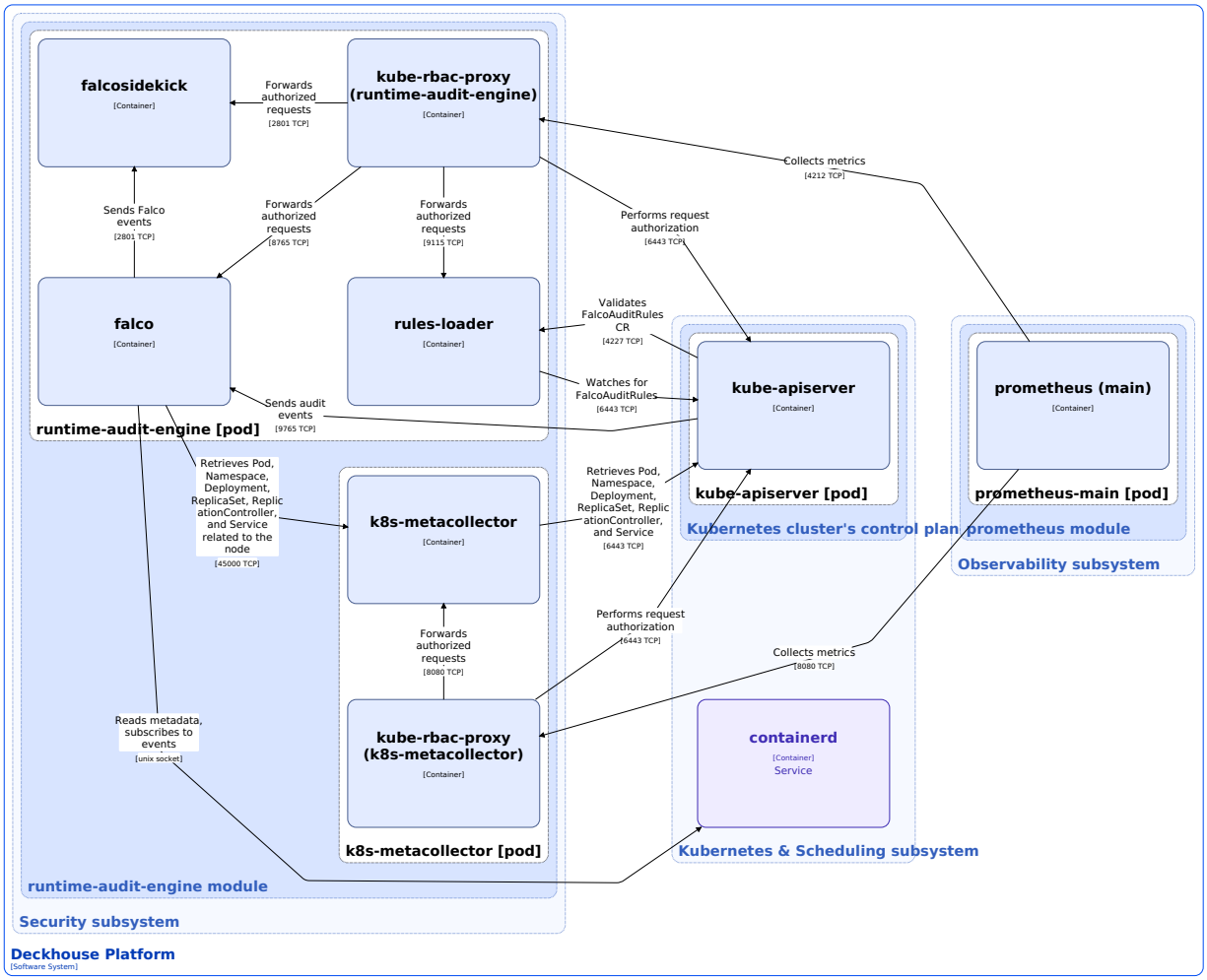
The `runtime-audit-engine` module implements runtime auditing in Deckhouse Platform (DP) based on the Falco threat detection system. The module collects Linux kernel events and Kubernetes API audit events (using the `k8saudit` plugin), enriches them with Kubernetes Pod metadata, and generates security events according to configured rules. Audit rules are defined using the FalcoAuditRules custom resource.

When the module is enabled, the `control-plane-configurator` ConfigMap is created in the `d8-runtime-audit-engine` namespace with the audit webhook URL and CA. The `control-plane-manager` module detects this ConfigMap and configures the control plane to send Kubernetes API audit events to the `runtime-audit-engine` module.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `runtime-audit-engine` module and its interactions with other DP components are shown in the following diagram:



Module components

The `runtime-audit-engine` module consists of the following components:

1. **Runtime-audit-engine** (DaemonSet): A component deployed on each cluster node. It collects audit events, evaluates rules, outputs triggered rules to stdout, and exports them as Prometheus metrics. Data is received from the Linux kernel via interception of system calls (syscalls) and from `containerd` via a Unix socket.

   The component includes the following containers:

   - **falco**: Main container that collects security events from cluster nodes and containerized applications in DP based on the Falco threat detection system.
   - **falcosidekick**: Sidecar container that receives events from the `falco` component and exports audit events as Prometheus metrics.
   - **rules-loader**: Sidecar container that performs the following operations:
     - Watches FalcoAuditRules custom resources and stores them in the shared `/etc/falco/rules.d/` Pod directory for processing by the `falco` component.
     - Validates the FalcoAuditRules custom resource.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC (Role-Based Access Control) that provides secure access to component metrics.

   > ⚠ The `falco` container has privileged access to each node operating system. The container security context includes the `BPF`, `SYS_RESOURCE`, `PERFMON`, `SYS_PTRACE`, and `SYS_ADMIN` capabilities.

2. **K8s-metacollector** (Deployment): A component that proxies requests to `kube-apiserver` to reduce control plane load. It also reduces the amount of metadata passed to `falco` by keeping only node-related data. K8s-metacollector collects metadata from `kube-apiserver` about Pod, Namespace, Deployment, ReplicaSet, ReplicationController, and Service resources.

   The component includes the following containers:

   - **k8s-metacollector**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to k8s-metacollector metrics.

Module interactions

The `runtime-audit-engine` module interacts with the following components:

1. **Kube-apiserver**:

    - Manages FalcoAuditRules custom resources.
    - Monitors Pod, Namespace, Deployment, ReplicaSet, ReplicationController, and Service resources.
    - Authorizes module component requests.

2. **Containerd**:

    - Provides container metadata.
    - Provides `containerd` events.

3. **Linux kernel**: Intercepts Linux kernel system calls (syscalls) in real time.

The following external components interact with the module:

1. **Kube-apiserver**:

    - Sends webhook requests to validate the FalcoAuditRules custom resource.
    - Sends audit events.

2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `control-plane-manager` documentation
- Module `runtime-audit-engine` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Secrets-store-integration module

The `secrets-store-integration` module delivers secrets to applications in Deckhouse Platform (DP) from an external store compatible with the HashiCorp Vault API.

The module provides the following capabilities:

- Delivers secrets to pods as mounted files or environment variables without storing them in etcd.
- Automatically connects to an internal Deckhouse Stronghold instance without manual configuration (`DiscoverLocalStronghold` mode).
- Works with any Vault-compatible secrets store in `Manual` mode.
- Automatically refreshes mounted secrets every two minutes when the value in the store changes.
- Injects an entrypoint for applications that cannot be modified to read secrets directly from the store.
- Delivers Base64-encoded binary secrets (for example, JKS keystores or Kerberos keytab files) with automatic decoding.

The operating mode (`Manual` or `DiscoverLocalStronghold`) is specified by the `settings.connectionConfiguration` parameter in the module settings.

The module works with the following custom resources:

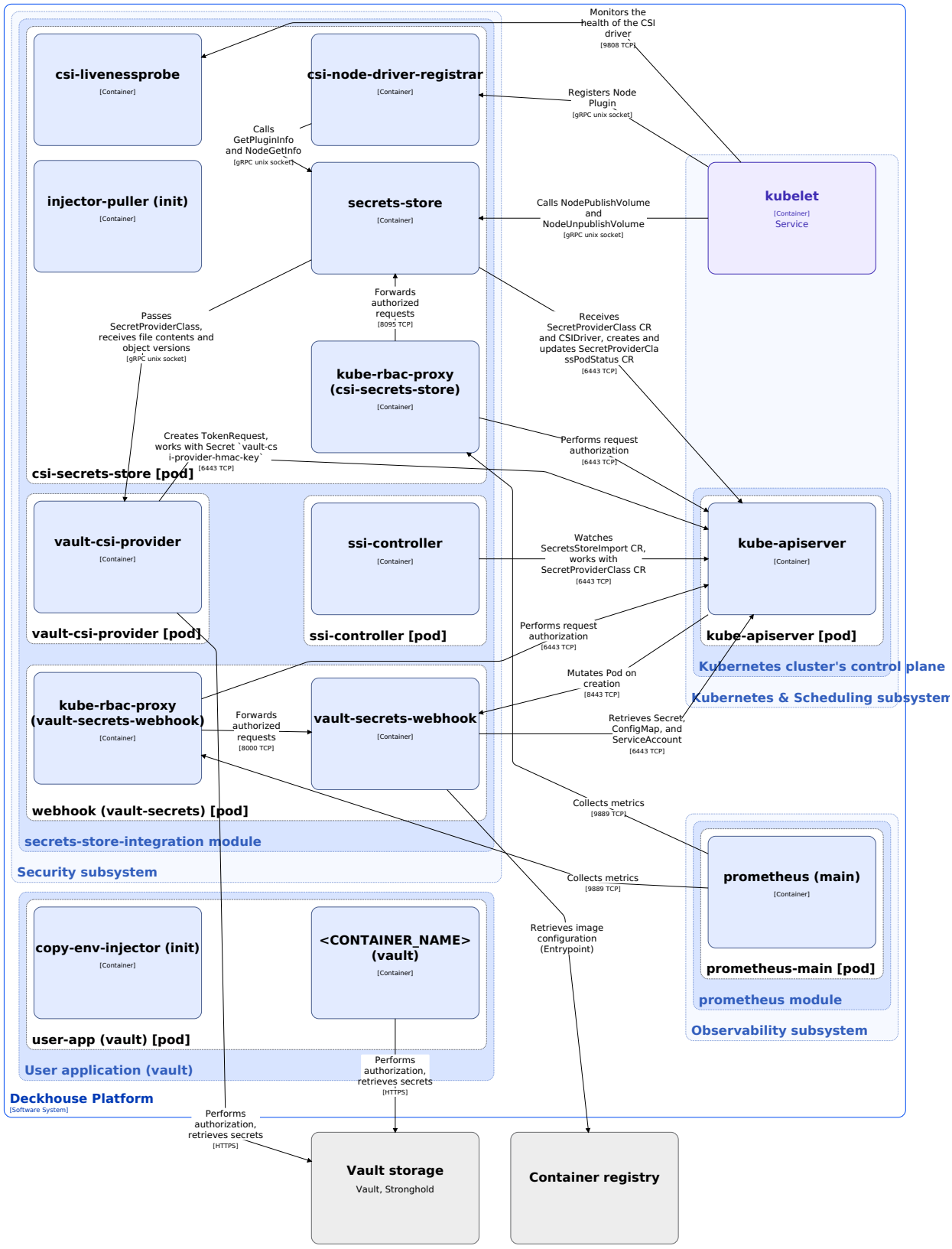
- SecretProviderClass: Describes which secrets to deliver to a pod and from which external store. The resource specification also defines connection parameters for the secrets source and path mappings in the container.
- SecretProviderClassPodStatus: Contains the status of the secret mounting process in a pod and diagnostic information.
- SecretsStoreImport: Stores the mapping of secrets between a Vault-compatible store and files in containers.

For more details about the module, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `secrets-store-integration` module and its interactions with other DP components are shown in the following diagram:



Module components

The `secrets-store-integration` module consists of the following components:

1. **Csi-secrets-store** (DaemonSet): Runs on all cluster nodes and implements the Container Storage Interface (CSI) standard to deliver secrets to a pod as mounted files.

   Csi-secrets-store performs the following actions:

   - Registers the ephemeral CSI driver `secrets-store.csi.deckhouse.io` in kubelet.
   - Interacts with vault-csi-provider to retrieve data from the secrets store.
   - Mounts secrets as files in a pod.
   - Rotates secrets.
   - Watches the Pod and CSIDriver resources and the SecretProviderClass custom resource.
   - Works with the SecretProviderClassPodStatus custom resource.

The component includes the following containers:

- **injector-puller**: Init container that performs a one-time launch of the injector binary (an executable that securely injects secrets from an external store into the application environment inside the pod) from the `secrets-store-integration/env-injector` service image. This one-time launch is necessary to preload the image onto every cluster node. The webhook component uses this image to deliver secrets to user applications as environment variables.
- **csi-node-driver-registrar**: Sidecar container that registers the CSI Node Plugin in kubelet. Calls the `GetPluginInfo` and `NodeGetInfo` RPCs in the secrets-store container to get plugin and node information.
- **secrets-store**: Main container.
- **csi-livenessprobe**: Sidecar container that monitors the CSI driver Unix socket and exposes the `/healthz` HTTP endpoint monitored by kubelet. If the `livenessProbe` check fails, kubelet restarts the csi-secrets-store pod.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to secrets-store container metrics.

2. **Vault-csi-provider** (DaemonSet): Runs on all cluster nodes and consists of a single **vault-csi-provider** container. Vault-csi-provider authenticates with the secrets store, retrieves data from it, and passes the data to csi-secrets-store.
3. **Ssi-controller** (Deployment): Consists of a single **ssi-controller** container. The controller watches SecretsStoreImport custom resources and creates SecretProviderClass custom resources based on them.
4. **Webhook** (Deployment): Implements mutating webhooks for Pod resources using Mutating Admission Controllers.

   The component mutates a pod if the pod has the `secrets-store.deckhouse.io/role` annotation. In this case, the component makes the following modifications to the pod manifest:

   - Adds an init container that copies a statically built injector binary from the `secrets-store-integration/env-injector` service image into a temporary directory shared by all containers in the pod.
   - If the pod manifest has the `secrets-store.deckhouse.io/env-from-path` annotation or a container uses secrets from the secrets store, then for each such container, including init containers, the component replaces the original startup command with a command that starts the injector binary.

     The injector receives the original startup command as an argument. If the container in the pod manifest has no startup command, the component retrieves the startup command from the image in the container registry.

   The component includes the following containers:

   - **vault-secrets-webhook**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to vault-secrets-webhook container metrics.

The following components are not part of the module, but the module affects them:

- **User-app**: Represents a typical user application that needs secrets delivered as environment variables.

  The component includes the following containers:

  - **copy-env-injector**: Init container added by the webhook component. It copies the injector binary from the `secrets-store-integration/env-injector` service image.
  - **<CONTAINER_NAME>**: One or more containers (including init containers) of the original user application whose startup command was changed by the webhook component to start the injector binary.

  The injector authenticates with the secrets store, retrieves data, passes secrets to the application as environment variables, and starts the original container command. If the `secrets-store.deckhouse.io/restart-on-secret-change` annotation is set to `watch-for-lease` or `watch-for-data`, the injector rotates secrets when they change in the store.

Module interactions

The `secrets-store-integration` module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests.
   - Retrieves CSIDriver, Secret, ConfigMap, and ServiceAccount.
   - Works with the SecretsStoreImport, SecretProviderClass, and SecretProviderClassPodStatus custom resources.
2. **Container registry**: Retrieves the startup command from the user container image.
3. **Secrets store**: Authorizes requests and retrieves secrets.

The following external components interact with the module:

1. **Kube-apiserver**: Invokes the mutating webhook when a Pod resource is created.
2. **Kubelet**:
   - Registers the Node Plugin.
   - Checks the CSI driver `livenessProbe`.
   - Calls the `NodePublishVolume` and `NodeUnpublishVolume` RPCs in the Node Plugin.
3. **Prometheus-main**: Collects metrics from the csi-secrets-store and webhook components.

Additional resources

- Module `secrets-store-integration` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Network subsystem

This subsection describes the architecture of the Network subsystem of Deckhouse Platform (DP).

The Network subsystem includes the following modules:

- `kube-dns`: Installs CoreDNS components for DNS management in the Kubernetes cluster.
- `node-local-dns`: Deploys a caching DNS server on each cluster node and exports DNS metrics to Prometheus for analyzing DNS operation in the cluster on the Grafana dashboard.
- `kube-proxy`: Manages the kube-proxy components responsible for networking and load balancing within the cluster.
- `cni-cilium`: Provides cluster networking using the CNI Cilium plugin.
- `cni-flannel`: Provides a network between multiple nodes in a cluster using the Flannel project.
- `cni-simple-bridge`: Provides basic networking for pods, with limited functionality compared to the platform's other CNI modules.
- `cilium-hubble`: Provides visualization of the cluster network stack if the Cilium CNI is enabled.
- `ingress-nginx`: Installs and manages the Ingress NGINX Controller using custom resources.
- `metallb`: Implements the LoadBalancer mechanism for Services in bare-metal clusters.
- `istio`: Implements a Service Mesh based on Istio for centralized management of network traffic in the cluster.
- `alb`: Implements an Application Load Balancer (ALB) based on the Kubernetes Gateway API.
- `sdn`: Provides software-defined networking (SDN) functions in the cluster — node network interface configuration, additional networks for pods and VMs, underlay networks, and system networks.
- `service-with-healthchecks`: Provides an internal load balancer with independent health checks for each port, unlike the standard Kubernetes Service load balancer.
- `static-routing-manager`: Manages static routes and `ip rule` entries on cluster nodes.

The subsection also describes:

- Cluster architecture with Istio enabled
- Application service architecture with Istio enabled

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Alb module

The `alb` module implements an application load balancer (ALB, Application Load Balancer) in Deckhouse Platform (DP) and lets you publish applications using the Kubernetes Gateway API. It deploys and configures the infrastructure for accepting and routing external requests, and validates the user's Gateway API configuration. ALB is based on the Envoy Open Source project.

The module works with the following custom resources of the `network.deckhouse.io` API group:

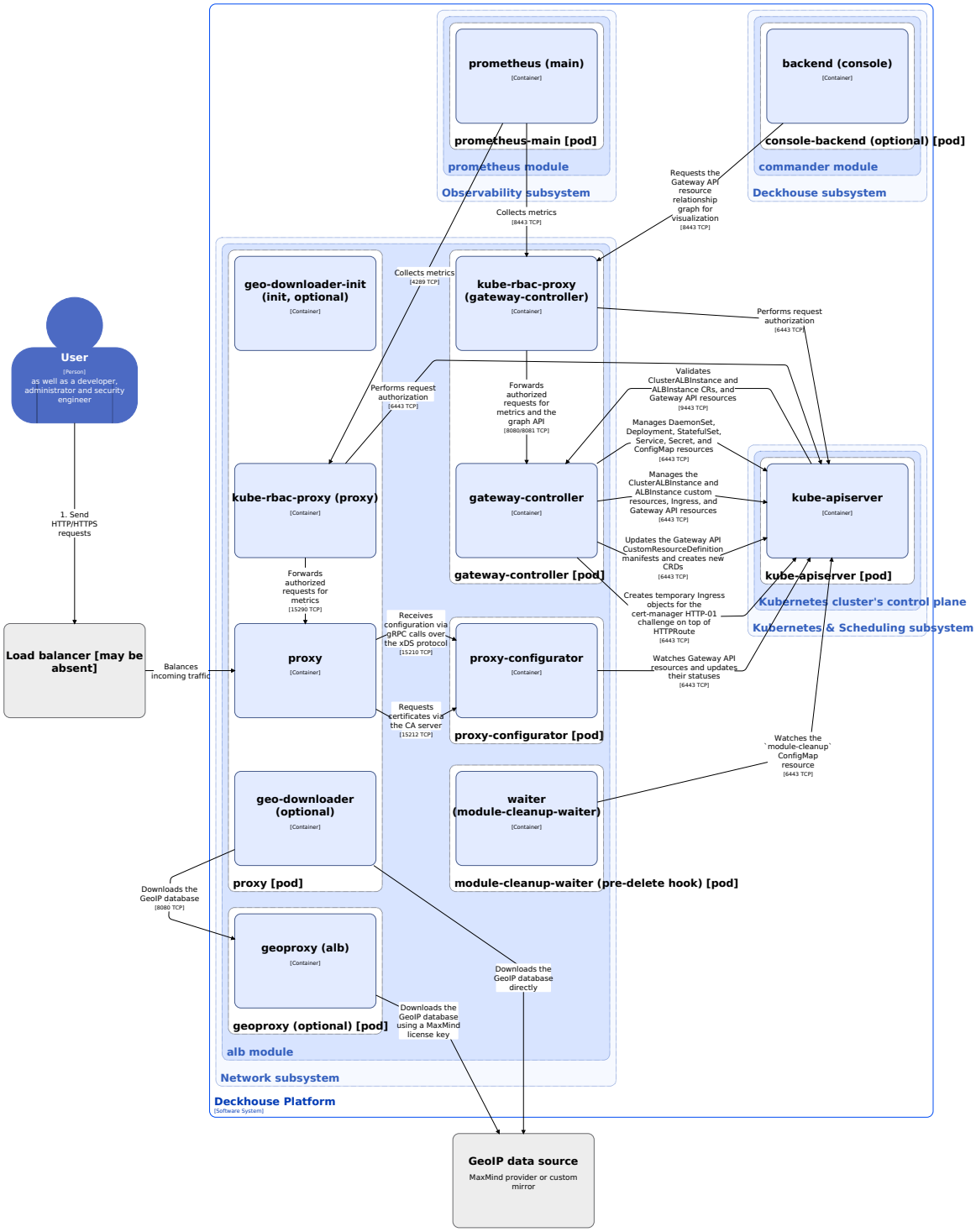
- ClusterALBInstance: Defines a cluster-wide ALB (data plane) instance, used for shared or platform-level Gateways.
- ALBInstance: Defines an ALB instance within a user namespace, used for application- or project-level Gateways.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `alb` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Gateway-controller** (Deployment): the module's central controller, which performs the following actions:
   - Manages the ClusterALBInstance and ALBInstance custom resources.
   - Installs Gateway API CRDs (the `gateway.networking.k8s.io` API group).
   - Implements the Gateway API controller for the `d8-alb` GatewayClass and creates a Gateway for every instance.
   - Validates the ClusterALBInstance and ALBInstance custom resources and the `gateway.networking.k8s.io` API group resources using the Validating Admission Controllers mechanism.
   - Creates temporary Ingress objects for the cert-manager HTTP-01 challenge on top of HTTPRoute when migrating from `ingress-nginx`.
   - Creates and removes the proxy and geoproxy components for every ClusterALBInstance/ALBInstance.
   - Provides a Graph API interface for retrieving the Gateway API resource relationship graph, used by the console module.

   It consists of the following containers:

   - **gateway-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to gateway-controller metrics and the graph API.

2. **Proxy-configurator** (Deployment): the Gateway API control-plane component, built from Istio Pilot (istiod) configured to work with Gateway API only (sidecar injection and istiod's built-in Gateway API infrastructure-provisioning mechanism are disabled).

   The component provides the following capabilities:

   - Implements the traffic management control plane.
   - Builds and validates the Envoy proxy configuration based on Gateway, HTTPRoute, GRPCRoute, TCPRoute, UDPRoute, TLSRoute, and ListenerSet resources.
   - Issues and renews certificates for the Envoy proxies using a built-in CA server.
   - Distributes the Envoy proxy configuration to the proxy components using the xDS protocol (a family of standardized API protocols for dynamically configuring data-plane components).
   - Updates the status of Gateway, HTTPRoute, GRPCRoute, TCPRoute, UDPRoute, TLSRoute, and ListenerSet resources.

   It consists of a single **proxy-configurator** container.

3. **Proxy** (Deployment or DaemonSet): an Envoy instance that implements the traffic management data plane, accepting and routing external traffic according to configuration received from proxy-configurator over the xDS protocol.

   The gateway-controller creates this component dynamically (not via a Helm template) for every ClusterALBInstance or ALBInstance custom resource. The workload kind depends on the resource type: a ClusterALBInstance is deployed as a DaemonSet, an ALBInstance as a Deployment behind a Service.

   It consists of the following containers:

   - **geo-downloader-init**: Optional init container that downloads the GeoIP database before Envoy starts.
   - **geo-downloader**: Optional sidecar container that periodically refreshes the local copy of the GeoIP database.
   - **proxy**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to the main container's metrics.

   The gateway-controller adds the first two containers only if the `spec.geoIP` parameter is set on the ClusterALBInstance/ALBInstance resource.

4. **Geoproxy** (StatefulSet): a caching proxy server for the proxy component that provides fast access to the GeoIP database downloaded from the MaxMind provider. The component also provides access to already-downloaded databases in clusters without internet access, which improves the stability of the Gateway API controller's work with GeoIP databases.

   The component provides the following benefits:

   - Saves MaxMind licenses (databases are downloaded from a single point once a day).
   - Persistent data storage (restarting components does not cause repeated requests to MaxMind servers).
   - The ability to specify your own mirror for downloading the databases.

   The gateway-controller creates this component programmatically for every ClusterALBInstance/ALBInstance, only if the `spec.geoIP.licenseKeySecretRef` parameter — a reference to the Secret holding the MaxMind license key — is set on the resource.

   It consists of a single **geoproxy** container.

5. **Module-cleanup-waiter** (Job): a Helm `pre-delete` hook of the module, launched by the Deckhouse controller before the `alb` module is removed. It waits for the gateway-controller to finish cleaning up the module's resources, and only then allows the release removal to complete. It does not take part in the module's normal operation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Installs resources of the `gateway.networking.k8s.io` API group.
   - Manages DaemonSet, Deployment, StatefulSet, Service, Secret, and ConfigMap resources.
   - Reads and updates Node and Namespace resources.
   - Manages the ClusterALBInstance and ALBInstance custom resources, Ingress resources, and Gateway API resources (the `gateway.networking.k8s.io` API group).
   - Authorizes requests for the gateway-controller component's metrics and graph API, and for the proxy component's metrics.

2. **GeoIP data source** (MaxMind provider or mirror): Downloads the GeoIP database.

The following external components interact with the module:

1. **Kube-apiserver**: Validates the ClusterALBInstance and ALBInstance custom resources and Gateway API resources (the `gateway.networking.k8s.io` API group).
2. **Prometheus-main**: Collects metrics from the gateway-controller and proxy components.
3. **Load balancer**: Balances incoming traffic between instances of the proxy component.
4. **Deckhouse web UI**: Requests the Gateway API resource relationship graph for visualization.

Additional resources

- Module `alb` documentation
- Module `console` documentation

- Module `ingress-nginx` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cni-cilium module

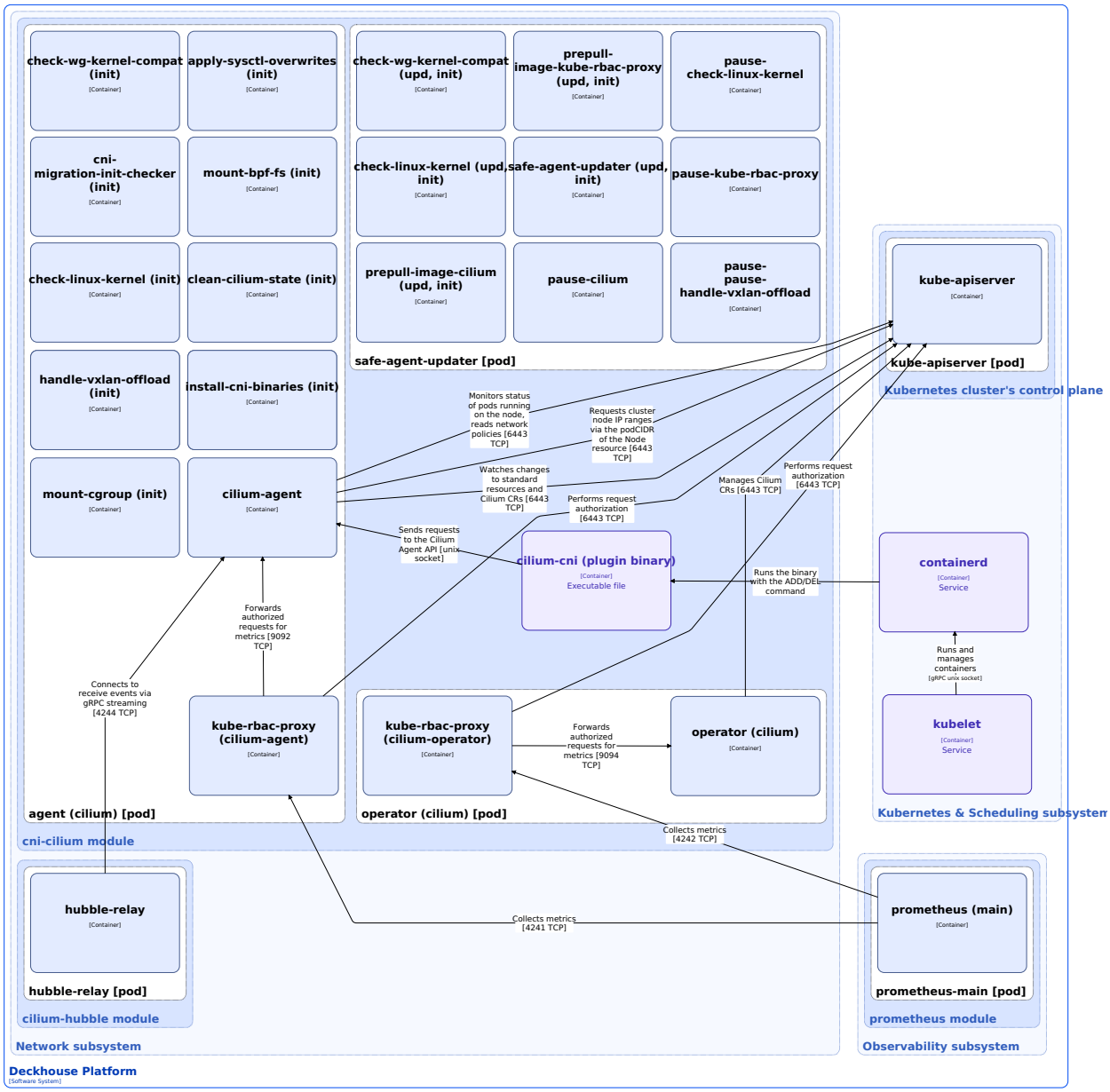
The `cni-cilium` module provides a network in a cluster. It is based on the Cilium project.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cni-cilium` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Operator** (Deployment): A component in the Cilium architecture that takes over the centralized management of global tasks in a Kubernetes cluster. Operator component performs the following functions:

   - **Custom resources management**: Automatically registers the Custom Resource Definitions (CRD) required for Cilium operation in the Kubernetes API, for example, CiliumEndpoint, CiliumEndpointSlice, CiliumNetworkPolicy. The full list of CRDs can be found in the Cilium documentation.
   - **Garbage collection for security identifiers**: Maintains a local cache of active identifiers (CiliumIdentity) and periodically scans it, deleting records about identifiers that have stopped signaling activity (heartbeat). This is important because identifiers are represented by a 16-bit integer, and running out of them (maximum 65536) can lead to problems.
   - **CiliumEndpointSlices (CES) Management**: Maintains up-to-date information about endpoints in the cluster, which is critical for the correct operation of network policies and load balancing.

   It consists of the following containers:

   - **operator**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to operator metrics. It is an open source project.

2. **Agent** (DaemonSet): Agent running on each node of the Kubernetes cluster. It is the main management component in the Cilium architecture. It is responsible for translating high-level configurations from the Kubernetes API into low-level settings for network interfaces and eBPF programs that actually handle traffic. Agent component performs the following functions:

   - **Endpoints management**: Monitors the status of all endpoints on its node. Each pod is assigned a unique IP

address and a security identity (numeric identifier), which is generated based on Kubernetes labels.

- **Network policies enforcement**: Calculates and applies network policies rules set in the cluster. Cilium uses the identity-based policies concept: the rules are based not on IP addresses, but on the security identity of the pod. The agent converts these rules into data structures (BPF maps) that the Linux kernel uses to filter traffic.
- **IP Address Management (IPAM)**: Allocates IP addresses for pods scheduled on its node using IP address ranges from the `spec.podCIDRs` or `spec.PodCidr` fields of the specification of the corresponding Node resource.
- **Datapath Orchestration**: Compiles, loads eBPF programs into the Linux kernel and binds them to network interfaces. It is these programs in the kernel that perform real packet processing: forwarding, policy filtering, connection tracking, NAT and load balancing.
- **Service Load Balancing**: Configures load balancing rules for services in a cluster using eBPF capabilities.
- **Cluster event handling**: The Agent works according to the event model: it constantly monitors changes in the Kubernetes API (creating pods, updating services, changing Endpoints, updating custom resources) and updates the configuration of eBPF programs on the fly.
- **Oservability**: The agent runs a local Hubble server (embedded in the agent process), which collects metrics and data on network flows. This allows you to monitor the network status, identify problems, and analyze traffic between pods. The Hubble server starts when the `cilium-hubble` module is enabled. The hubble-relay component of the `cilium-hubble` module connects to Cilium agents to receive events via a gRPC streaming.

How it works: when an event occurs in the cluster (for example, a new pod starts), the agent receives a notification about it and, based on the current cluster status and policies, configures eBPF programs so that traffic for this pod passes correctly.

It consists of the following containers:

- **cni-migration-init-checker**: Init container waiting for the migration process from another CNI plugin to complete, if such a migration has been started.
- **check-wg-kernel-compat**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with WireGuard used in CNI Cilium.
- **check-linux-kernel**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with CNI Cilium.
- **clearing-unnecessary-iptables**: Init container running script for cleaning up unnecessary `iptables` rules.
- **handle-vxlan-offload**: Init container correcting UDP segmentation settings (if tunnel mode is set to `VXLAN`).
- **config**: Init container generating configuration file used to configure the agent.
- **mount-cgroup**: Init container configuring Linux cgroups (control groups).
- **apply-sysctl-overwrites**: Init container configuring the Linux kernel parameters.
- **mount-bpf-fs**: Init container mounting the BPF file system.
- **clean-cilium-state**: Init container running a script for clearing the CNI Cilium state.
- **install-cni-binaries**: Init container running a script for installing binary files used by the Agent.
- **agent**: Main container.
- **kube-rbac-proxy**: Sidecar container providing authorized access to agent metrics (described above).

3. **Cilium-cni**: Executable file run by the containerd component that passes certain commands as an argument in accordance with the CNI specification, for example, ADD when starting the container and DEL when deleting it. Cilium-cni executable interacts with the Cilium agent API via a Unix socket and initiates a datapath configuration to provide network connectivity, load balancing, and network policies for pod. Datapath in Cilium is a component that runs in the Linux kernel and is responsible for real low-level processing of network packets in a Kubernetes cluster. Simply put, this is the “road” along which data travels: how packets travel from one pod to another, how they are routed, how network policies and load balancing are applied.
4. **Safe-agent-updater** (DaemonSet): A special application designed to eliminate disruptive situations that may occur when updating the version of the Cilium agent and lead to problems with the network availability of DP components.

   It consists of the following containers:

   - **check-wg-kernel-compat**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with WireGuard used in CNI Cilium.
   - **check-linux-kernel**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with CNI Cilium.
   - A set of init containers for prepulling new versions of images of the corresponding containers of the agent component:
     - **prepull-image-cilium**.
     - **prepull-image-kube-rbac-proxy**.
   - **safe-agent-updater**: Init container that compares the values of special annotations in the DaemonSet manifest of the updated Cilium agent with the values of the corresponding annotations in the metadata of the agent pod running on the node. In particular, the `safe-agent-updater-daemonset-generation` annotation stores the hash sum of the agent image. If the hash sums do not match, safe-agent-updater deletes the running pod and waits until the pod with the new agent version enters the `Ready` state.
   - A set of sidecar containers for prepulling new versions of images of the corresponding containers of the agent component. Containers are on pause and perform only the function of storing images:
     - **pause-cilium**.
     - **pause-check-linux-kernel**.
     - **pause-kube-rbac-proxy**.
     - **pause-pause-handle-vxlan-offload**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages Cilium custom resources.
   - Watches changes to standard Kubernetes API resources and Cilium custom resources.
   - Monitors status of pods running on the node, reads network policies.
   - Requests the IP ranges of cluster nodes via the `PodCidr`/`podCIDRs` attributes of the Node resource specification.
   - Authorizes requests for agent and operator metrics.

The following external components interact with the module:

1. **Hubble-relay**: Connects to Cilium agents to receive events via gRPC streaming.
2. **Prometheus-main**: Collects metrics from agent and operator.
3. **Containerd**: Runs the cilium-cni executable file with certain commands according to the CNI specification, for example, ADD when starting the container and DEL when deleting it.

Additional resources

- Module `cilium-hubble` documentation
- Module `cni-cilium` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cni-flannel module

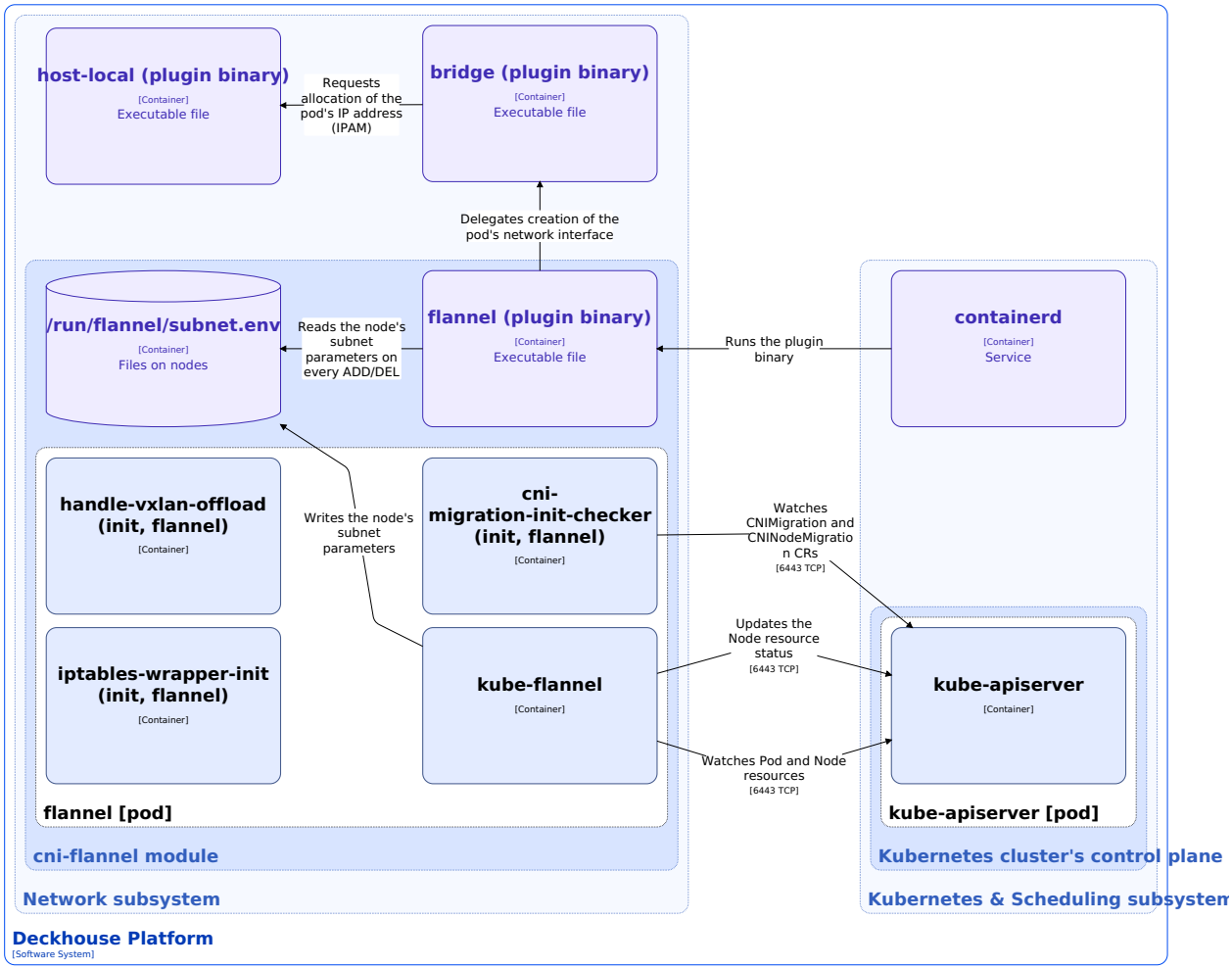
The `cni-flannel` module provides a network between nodes in a Deckhouse Platform cluster. It is based on the Flannel project and implements the Container Network Interface (CNI).

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cni-flannel` module and its interactions with other components of platform are shown in the following diagram:



Module components

The module consists of the following components:

1. **Flannel** (DaemonSet): Runs on every cluster node and implements Flannel's Layer 3 overlay network between nodes.

   Depending on the `podNetworkMode` parameter, it uses the host-gw or VXLAN backend.

   The component takes the pod subnet from the `spec.podCIDR` field of the Node resource and writes the resulting subnet parameters (network, subnet, MTU) to the `/run/flannel/subnet.env` file on the node. The Flannel CNI plugin uses this file on every pod creation or removal.

   It consists of the following containers:

   - **cni-migration-init-checker**: Init container waiting for the migration process from another CNI plugin to complete, if such a migration has been started.
   - **iptables-wrapper-init**: Init container that prepares executables required for working with iptables.
   - **handle-vxlan-offload**: Optional init container that corrects UDP segmentation settings on the node's network interface. It is created by the Deckhouse controller if the `podNetworkMode` parameter is set to `VXLAN`.
   - **kube-flannel**: Main container.

2. **Flannel (CNI plugin)**: An executable file run by the containerd component according to the CNI specification, for example ADD when creating a pod's network namespace and DEL when removing it.

   On startup, the component reads the subnet parameters from the `/run/flannel/subnet.env` file and delegates creation of the pod's network interface to the `bridge` CNI plugin, which in turn requests an IP address from the `host-local` IP Address Management (IPAM) plugin. Both `bridge` and `host-local` are standard Open Source CNI plugins, installed on every cluster node independently of this module, as part of kubelet installation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches CNIMigration and CNINodeMigration custom resources, and Pod and Node resources.
   - Updates the status of the Node resource.

2. **Bridge (CNI plugin)**: Creates the pod's network interface.

The following external components interact with the module:

1. **Containerd**: Runs the flannel executable file with certain commands according to the CNI specification, for example, ADD when creating a pod’s network namespace and DEL when removing it.

Additional resources

- Global custom resources
- Module `cni-flannel` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cni-simple-bridge module

The `cni-simple-bridge` module provides basic networking for pods in Deckhouse Platform and implements the Container Network Interface (CNI).

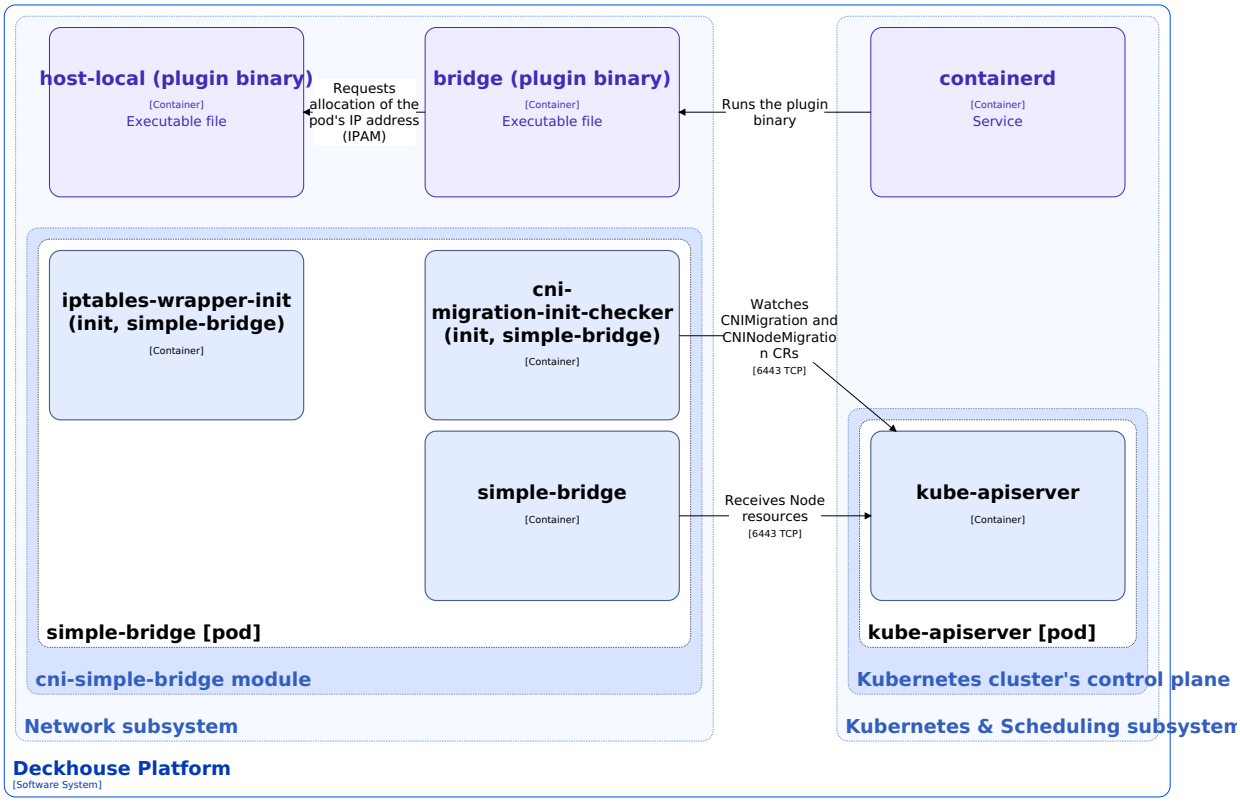
> The `cni-simple-bridge` CNI plugin was used by default for the `cloud-provider-aws`, `cloud-provider-azure`, `cloud-provider-gcp`, and `cloud-provider-yandex` modules in Deckhouse platform versions earlier than 1.77.0. Starting with platform 1.77.0, these modules use Cilium CNI provided by the `cni-cilium` module by default.

The module has no configurable parameters.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cni-simple-bridge` module and its interactions with other components of Deckhouse Platform are shown in the following diagram:



Module components

The module consists of the following components:

1. **Simple-bridge** (DaemonSet): Runs on every cluster node and prepares the CNI plugin for operation. To do this, it obtains the pod subnet from the `spec.podCIDR` field of the Node resource and builds the configuration for running the `bridge` executable, which containerd invokes on every creation or removal of a pod's network namespace. When invoked, `bridge` requests an IP address from the `host-local` IP Address Management (IPAM) plugin.

   `bridge` and `host-local` are standard Open Source CNI plugins, installed on every cluster node independently of this module, as part of kubelet installation.

   It consists of the following containers:

   - **cni-migration-init-checker**: Init container waiting for the migration process from another CNI plugin to complete, if such a migration has been started.
   - **iptables-wrapper**: Init container that prepares executables required for working with iptables.
   - **simple-bridge**: Main container.

Module interactions

The module interacts with the **Kube-apiserver** component:

- Reads Node resources.
- Watches CNIMigration and CNINodeMigration custom resources.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `cloud-provider-azure` documentation
- Module `cloud-provider-gcp` documentation
- Module `cloud-provider-yandex` documentation
- Module `cni-cilium` documentation
- Module `cni-simple-bridge` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cilium-hubble module

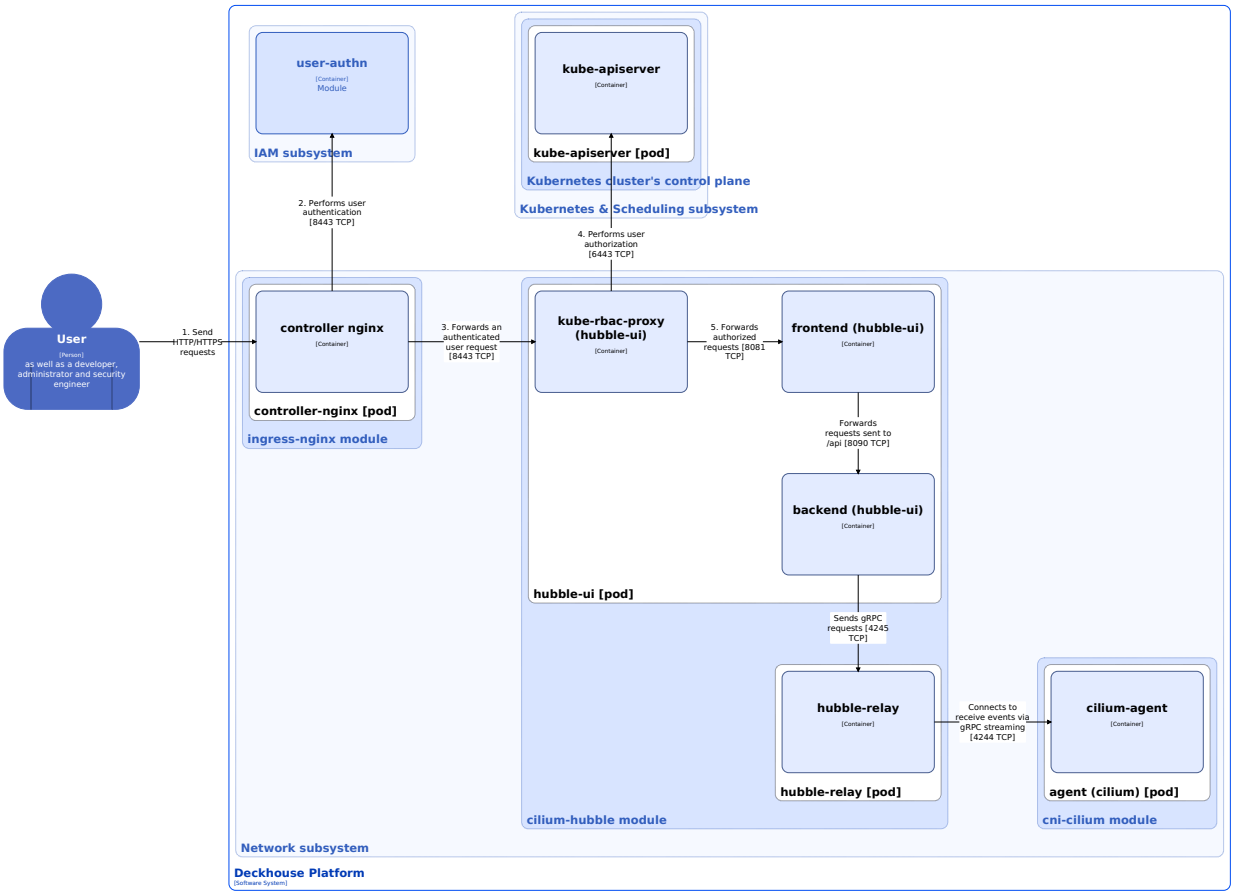
The `cilium-hubble` module provides visualization of the cluster network stack if the Cilium CNI is enabled.

For more details about module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cilium-hubble` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



> **i** The numbers in the diagram indicate the sequence of steps that a user request goes through before reaching the hubble-ui component:
>
> - In steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed using the `user-authn` module;
> - In steps 4 and 5, the user is authorized based on Kubernetes RBAC to provide secure access.

Module components

The module consists of the following components:

1. **Hubble-relay**: Component that aggregates events from all cluster nodes into a single view. Hubble-relay establishes a permanent connection to each Cilium Agent on the nodes and, through the gRPC stream, receives events, deduplicates them, and provides a single gRPC endpoint for the Hubble CLI and Hubble UI (web interface). While doing this, hubble-relay does not store history, it broadcasts an event stream in real time.

   It consists of a single container:

   - **hubble-relay**: Main container. It is a part of Cilium project.

2. **Hubble-ui**: Component that implements the web interface. Hubble-ui connects to hubble-relay and provides convenient access to data that comes from Cilium agents. Hubble-ui builds an interactive Service Map and a flow table with filters.

   It consists of the following containers:

   - **frontend**: Container that is an NGINX proxy server, which distributes static files of the hubble-ui web interface and forwards requests to the `/api` endpoint to the **backend** container of the component.
   - **backend**: Container implementing the hubble-ui API that sends requests to the hubble-relay component to receive in real time data coming from Cilium agents.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to Hubble UI. It is an open source project.

   Frontend and backend containers are built based on Hubble-UI, which is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Cilium-agent**: Receives in real time an event stream from Cilium agents.
2. **Kube-apiserver**: Authorizes requests to Hubble UI.

The following external components interact with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `cilium-hubble` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Ingress-nginx module

The `ingress-nginx` module installs and manages the Ingress NGINX Controller using the IngressNginxController custom resource.

The module can operate in high availability (HA) mode and provides flexible configuration for placing Ingress controllers on cluster nodes, as well as tuning controller behavior according to the specifics of the underlying infrastructure.

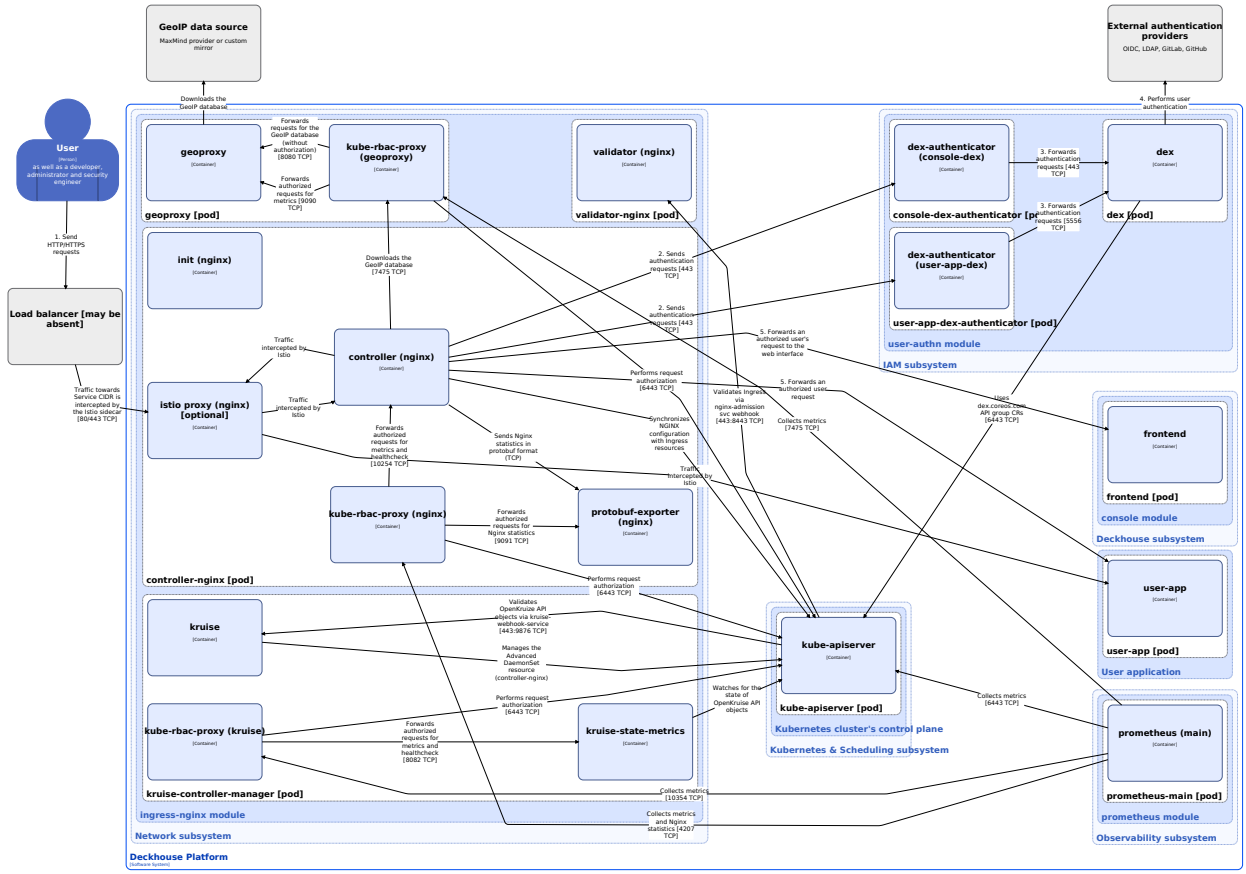
The module supports running and configuring multiple instances of the Ingress NGINX Controller independently. This allows, for example, separating external and intranet application Ingress resources.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `ingress-nginx` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller-nginx** (Advanced DaemonSet): Non-standard DaemonSet with advanced capabilities managed by kruise-controller-manager.

It consists of the following containers:

- **init**: Init container configuring a file system for the Ingress controller.
- **controller**: Main container running Ingress NGINX Controller and implementing the core module logic. It is an open source project.
- **protobuf-exporter**: Sidecar container in the ingress-controller pod that receives NGINX statistics in protobuf format. It parses and aggregates the messages according to predefined rules and exports metrics in Prometheus format. This exporter is developed by Flant.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to controller metrics and status as well as to `protobuf-exporter`. It is an open source project.
- **istio-proxy**: Istio sidecar container added to the pod when the `spec.enableIstioSidecar` parameter of the IngressNginxController custom resource is enabled. In this case, part of the user traffic passes through it.

2. **Validator-nginx** (Deployment): Consists of a single container. Validator is an Ingress NGINX Controller running in validation mode with a reduced set of privileges. It implements a webhook server used to validate Ingress resources via the Validating Admission Controllers mechanism.

3. **Kruise-controller-manager** (Deployment): Controller that manages the Advanced DaemonSet custom resource. This DaemonSet extension provides advanced update capabilities for the Ingress NGINX Controller that are not available in the standard Kubernetes DaemonSet controller.

   It consists of the following containers:

   - **kruise**: Main container running kruise-controller-manager.
   - **kruise-state-metrics**: Sidecar container that monitors the state of OpenKruise API objects and exposes corresponding metrics (but not metrics of the kruise-controller-manager itself).
   - **kube-rbac-proxy**: Sidecar container providing authorized access to controller metrics and status (described above).

4. **Geoproxy** (StatefulSet): Caching proxy server for the Ingress controller that provides quick access to the GeoIP database downloaded from the MaxMind provider. This component also grants access to already downloaded bases in clusters that have no internet access, improving the stability of the Ingress NGINX Controller when working with GeoIP databases.

   This component provides the following features:

   - MaxMind license saving (databases are downloaded from a single point once a day).
   - Persistent data storage (if components are restarted, it doesn't require accessing the MaxMind servers again).
   - It lets you specify a custom mirror for downloading databases.

   It consists of the following containers:

   - **geoproxy**: Proxy server. This server is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to geoproxy metrics (the GeoIP databases are available without authorization). This container is described in detail above.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Synchronizes the NGINX configuration when Ingress resources change.
   - Authorizes requests for metrics, statistics, and controller status checks.
   - Forwards external HTTP requests to the Kubernetes API endpoint.

2. **GeoIP database source** (MaxMind provider or custom mirror): The module downloads the GeoIP database.

3. **Dex-authenticator of platform services and user applications**: Used to authenticate requests in dex via dex-authenticator, which acts as an OAuth2 proxy.

4. **DP platform services** (such as `console`, `dashboard`, Grafana, and others): The module forwards HTTP requests that have been authenticated via Dex.

5. **User services deployed in DP**: The module forwards external HTTP requests to user services. To enable this, the user must create the corresponding Ingress resources and, if authentication via Dex is required, the DexAuthenticator custom resource.

> **i** To keep the diagram simple, it shows interactions between ingress-controller and only one DP service, the frontend component of the `console` module and its corresponding console-dex-authenticator.

The following external components interact with the module:

1. **Kube-apiserver**: Uses a validation webhook to validate created or updated Ingress resources.
2. **Prometheus-main**: Collects metrics from ingress and kruise controllers, geoproxy, as well as NGINX statistics.
3. **Load balancer**: Distributes HTTP/HTTPS traffic across healthy ingress-controller instances.

Receiving traffic from external networks

Methods for receiving traffic from external networks are described in the `spec.inlet` parameter of the IngressNginxController custom resource.

For inlet types LoadBalancer, LoadBalancerWithProxyProtocol, and LoadBalancerWithSSLPassthrough, the load balancer shown in the diagram is automatically provided by the cloud provider (when DP is deployed in a cloud environment) or can be implemented using the MetalLB controller (for bare-metal installations). For configuration details, refer to the

`metallb` module documentation.

For inlet types HostPort, HostPortWithProxyProtocol, HostPortWithSSLPassthrough, and HostWithFailover, the load balancer is deployed by the user or may be absent. In this case, the user must configure the load balancer backends or otherwise ensure network connectivity to the ingress-controller. The ingress-controller entry point in this case is the ports on the cluster nodes where the controller is running.

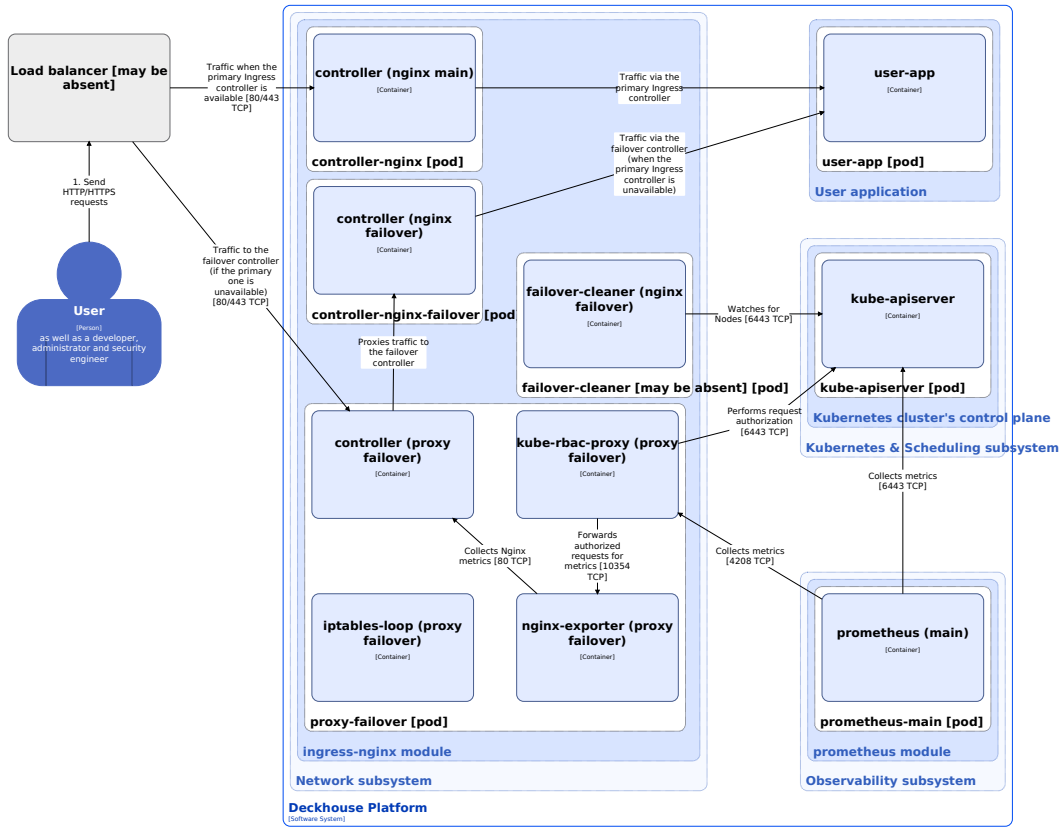
Architecture of Ingress controller with HostWithFailover inlet type

When the value `HostWithFailover` of the `spec.inlet` parameter of the IngressNginxController custom resource is set, two Ingress controllers are installed in the cluster: the primary controller and the failover controller, as well as a proxy-failover controller that coordinates traffic switch between them.

The primary controller starts in `hostNetwork`, while the failover controller starts in `podNetwork`. If the main controller pod is unavailable on the node, the proxy failover starts proxying traffic to the failover controller pod using the `PROXY PROTOCOL` to save information about the client's IP address.

> **i** The following diagram does not show the architecture of the main Ingress controller, as well as the module interactions, as they are described in detail in the diagram above.

The Level 2 C4 architecture of Ingress controller with HostWithFailover inlet type and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Components of failover Ingress controller

1. **Controller-nginx-failover** (Advanced DaemonSet): Failover Ingress-controller deployed on the same nodes as the main one. Failover controller has the same containers on the pod with the same purpose as the main Ingress controller.
2. **Proxy-failover** (Advanced DaemonSet): Proxy server.

   It consists of the following containers:

   - **controller**: Server that performs the following functions:
     - Monitors a performance of the main Ingress controller and returns its status.
     - Monitors the NGINX configuration file on the node and reboots Ingress controller if this configuration is changed.
     - Starts a NGINX instance that proxies traffic to the failover controller in case the main one is unavailable. Traffic is switched to the failover controller and back to the main controller with the use of `iptables` rules, that are configured by the iptables-loop container depending on the availability of the `80` and `443` TCP ports on the main controller.

     This controller is developed by Flant.
   - **iptables-loop**: Sidecar container that updates the `iptables` rules on the node that are necessary for the

Ingress controller to work. It is developed by Flant.

- **nginx-exporter**: Sidecar container that connects to NGINX over HTTP and exports metrics in the Prometheus format. It is an open source project.
- **kube-rbac-proxy**: Sidecar container providing authorized access to controller metrics and status (described above).

3. **Failover-cleaner** (DaemonSet): Component deployed to the cluster nodes that are labeled with `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true`. It cleans up the `iptables` rules. If the Ingress controller operates normally, the failover-cleaner component doesn't run on any node.

Failover Ingress controller interactions

Failover Ingress controller interacts with the following components:

1. **Kube-apiserver**:
   - Watches Node resources for `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true` labels.
   - Authorizes requests for metrics, statistics, and controller status checks.
   - Forwards external HTTP requests to the Kubernetes API endpoint.
2. **GeoIP database source** (MaxMind provider or custom mirror): Failover Ingress controller downloads the GeoIP database.
3. **Dex-authenticator of platform services and user applications**: Used to authenticate requests in dex via dex-authenticator, which acts as an OAuth2 proxy.
4. **DP platform services** (such as `console`, `dashboard`, Grafana, and others): The module forwards HTTP requests that have been authenticated via Dex.
5. **User services deployed in DP**: The module forwards external HTTP requests to user services. To enable this, the user must create the corresponding Ingress resources and, if authentication via Dex is required, the DexAuthenticator custom resource.

The following external components interact with the failover Ingress controller:

1. **Prometheus-main**: Collects failover Ingress controller metrics.
2. **Load balancer**: Routes HTTP/HTTPS traffic in case the main Ingress controller is unavailable.

Additional resources

- Module `ingress-nginx` documentation
- Module `metallb` parameters
- Module `user-authn` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Istio module

The `istio` module implements a Service Mesh based on Istio for centralized management of network traffic in Deckhouse Platform (DP). The module provides mTLS (Mutual Transport Layer Security), request authorization, traffic routing, load balancing, and observability of interactions between applications.

The `istio` module allows several Istio versions to run at the same time. The module's `globalVersion` parameter specifies which Istio version is used by default for namespaces labeled `istio-injection: enabled`. If a namespace needs to use an Istio version other than the default one, it is labeled with the label corresponding to the Istio revision instead, for example, `istio.io/rev: v1x27`.

The module works with the following custom resources.

Resources managed directly by the module (the `deckhouse.io` group):

- IngressIstioController: Describes an Istio ingress gateway instance serving the selected gateway class.
- IstioFederation: Marks one or more remote clusters as trusted for mesh federation (available in the Enterprise Edition and Ultimate).
- IstioMulticluster: Marks one or more remote clusters as trusted for a multicluster configuration (available in the Enterprise Edition and Ultimate).
- WaypointInstance: Describes a waypoint ambient proxy created by the waypoint-controller component (available in the Enterprise Edition and Ultimate).

The module also installs and uses the standard Istio custom resources (the `networking.istio.io`, `security.istio.io`, `telemetry.istio.io`, and `extensions.istio.io` groups — VirtualService, DestinationRule, Gateway, PeerAuthentication, and others). For more details, see the Istio custom resource reference.

> ⚠ DP only supports operator-based management for Istio version 1.25. All later versions run without the operator. Istio 1.25 is deprecated and will be removed in a future update.

If an Istio version with operator support is requested, the following Sail Operator custom resources of the `sailoperator.io` group are installed and used:

- Istio: Represents an Istio Service Mesh deployment consisting of one or more control planes.
- IstioRevision: Represents a single revision of the Istio control plane.

The set of module components and its architecture depend on the DP edition. The Enterprise Edition (EE) and Ultimate adds cross-cluster service mesh federation, periodic service mesh configuration analysis, and support for Istio ambient mode.

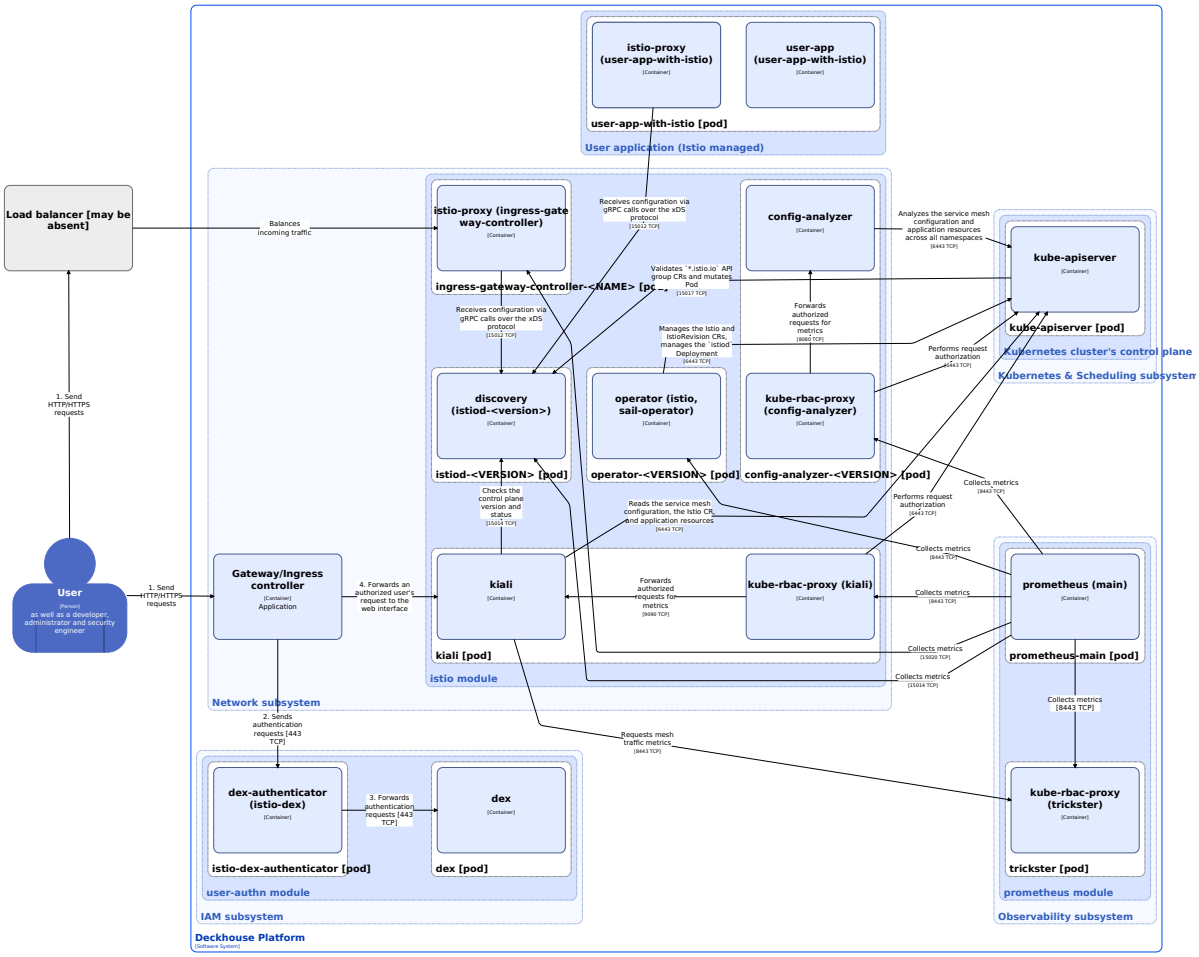
For more details about module configuration, refer to the corresponding documentation section.

Module architecture

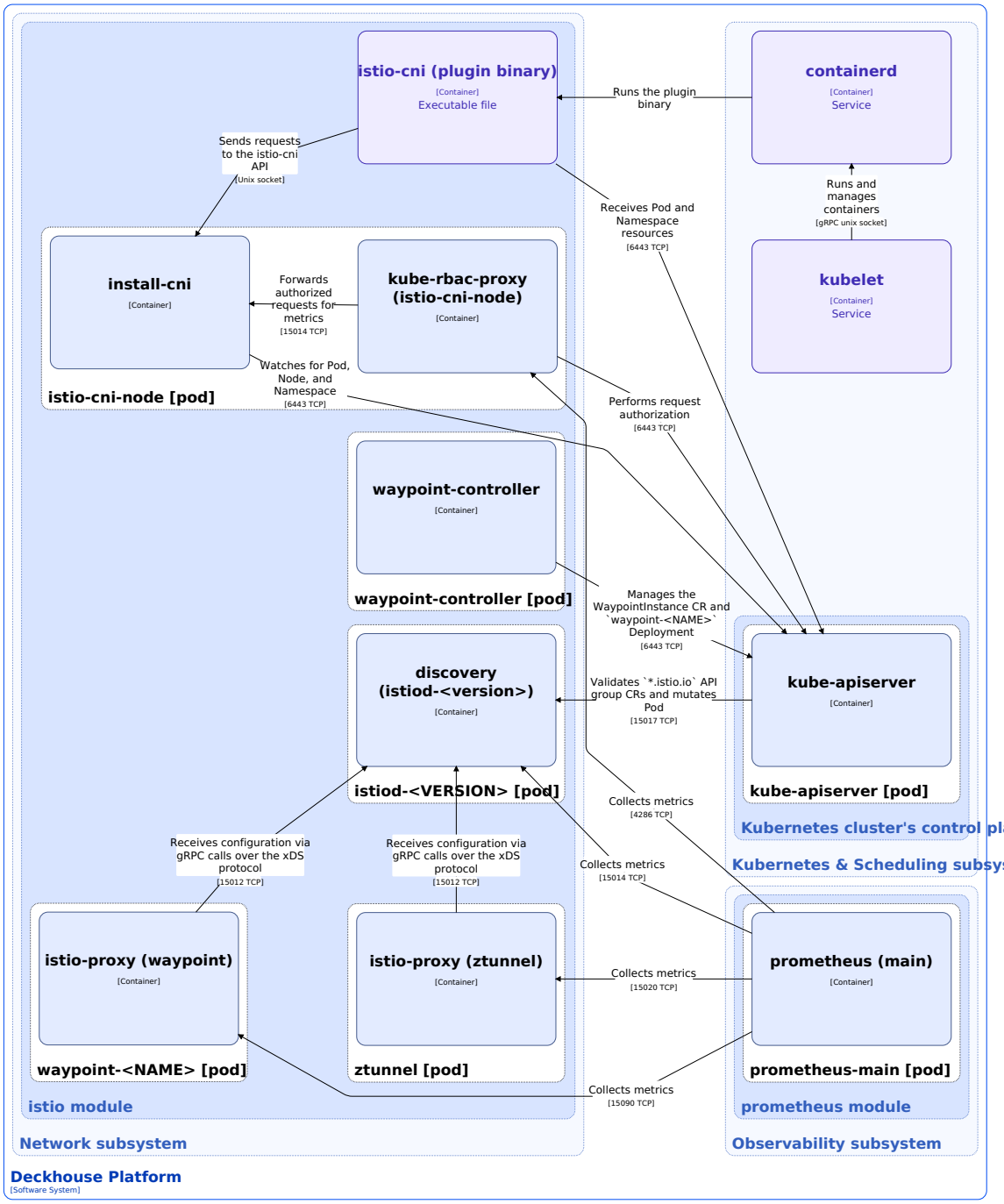
The Level 2 C4 architecture of the `istio` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams:

> **i** The following simplifications are made in the diagrams:
>
> - The diagrams show containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagrams.

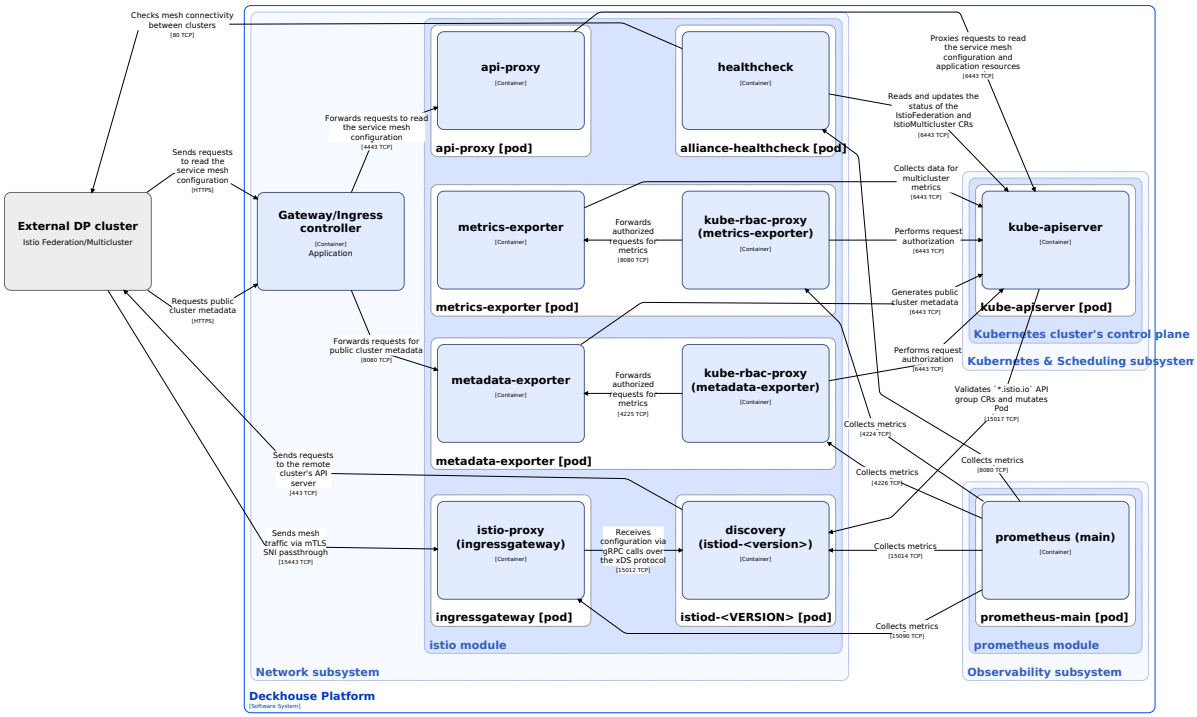
- Base module functionality (control plane, CNI, ingress gateway, Kiali, config-analyzer):



- Ambient mode (ztunnel, waypoint-controller; available only in the Enterprise Edition and Ultimate, disabled by default — the diagram shows only the differences from the base configuration):



- Federation and multicluster configuration (available only in the Enterprise Edition and Ultimate, disabled by default — the diagram shows only the differences from the base configuration):



Module components

Main module components

The module consists of the following components:

1. **Operator-<VERSION>** (Deployment): an implementation of the Sail Operator that manages the Istio control plane lifecycle. The component is responsible for installing all the resources required for a given control plane version to work.

   DP only supports the operator for Istio version 1.25.

   The component watches the Istio and IstioRevision custom resources and uses them to create the `istiod-<VERSION>` Deployment, a Service, and a ConfigMap. Webhook management for validation and mutation is deliberately disabled in the operator; it is handled by the Deckhouse controller of the `deckhouse` module when the module's Helm chart is applied.

   It consists of a single container:

   - **operator**: Main container.

2. **Istiod-<VERSION>** (Deployment): the Istio control plane component that:

   - Distributes sidecar proxy routing configuration via xDS.
   - Issues certificates for workloads managed by Istio.
   - Validates user application pods and mutates them to inject sidecar containers, via Validating/Mutating Admission Controllers.
   - Validates custom resources of the `*.istio.io` API groups via Validating Admission Controllers.

   For Istio 1.25 (deprecated and scheduled for removal), the component is created and managed by the operator-<VERSION> component through the Istio and IstioRevision custom resources. For other Istio versions supported in DP, it is deployed directly by the module's Helm chart.

   It consists of a single container:

   - **discovery**: Main container.

3. **Ingress-gateway-controller-<NAME>** (DaemonSet): controller that handles incoming mesh traffic from applications. Created by the Deckhouse controller of the `deckhouse` module for each IngressIstioController custom resource. The `<NAME>` placeholder is the name of the IngressIstioController resource.

   It consists of a single container:

   - **istio-proxy**: Main container based on the open-source Envoy project, which handles incoming traffic and receives configuration via xDS from the istiod controller.

4. **Kiali** (Deployment): Kiali web interface for managing and observing Istio resources and user services running under Istio, allowing you to:

   - Visualize connections between services.
   - Diagnose problematic connections between services.
   - Diagnose the state of the Istio control plane.

   It consists of the following containers:

   - **kiali**: Main container.

- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to the Kiali web interface.

Authentication of Kiali web interface users is performed by the `user-authn` module through a dedicated dex-authenticator.

5. **Istio-config-analyzer-<VERSION>** (Deployment): component that performs periodic service mesh configuration analysis (`istioctl analyze`) and exports the analysis results as Prometheus metrics.

   The component is created by the Deckhouse controller of the `deckhouse` module for each Istio revision if the `.settings.configAnalysis.enabled` module parameter is set to `true` (enabled by default).

   It consists of the following containers:

   - **istio-config-analyzer**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to metrics.

Ambient mode components

The following additional components are created when Istio runs in ambient mode:

1. **Istio-cni-node** (DaemonSet): Istio component that installs the CNI plugin on each cluster node and, in Istio ambient mode, sets up traffic interception for pods.

   The component prepares the `istio-cni` binary and appends it as an additional plugin to the first CNI config found in the `/etc/cni/net.d/` directory on each cluster node. In the standard DP configuration this is the `05-cilium.conflist` file, created by the Cilium CNI plugin of the `cni-cilium` module. When creating each pod, kubelet (via containerd) calls both CNI plugins in sequence — first cilium, then istio-cni. The result produced by the first plugin is passed to the second.

   In Istio ambient mode, the component handles API requests from the `istio-cni` CNI plugin and configures routing to the ztunnel component.

   The component is created by the Deckhouse controller if the `.settings.dataPlane.trafficRedirectionSetupMode` module parameter is set to `CNIPlugin` (the default is `InitContainer`).

   It consists of the following containers:

   - **install-cni**: Main container that installs and configures the CNI plugin on the node.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to install-cni metrics.

2. **Istio-cni**: binary invoked by containerd, which receives the command (for example, ADD when a container starts and DEL when it is removed) and other parameters via environment variables, per the CNI specification, and the JSON configuration via stdin.

   On each invocation, istio-cni performs the following actions:

   - Processes the input from containerd: the JSON configuration, and the pod's name and namespace.
   - Retrieves pod and namespace information from kube-apiserver.
   - Stops processing if the namespace is in `exclude_namespaces` (a configuration parameter in the `cni-config` ConfigMap).
   - In ambient mode, checks whether the pod is enabled for ambient via labels and, if so, notifies the istio-cni-node component via a Unix socket.
   - In sidecar mode, checks that the pod has no `istio-init` container, has an `istio-proxy` container, and checks annotations; if the conditions are met, it enters the pod's network namespace (netns) and sets up interception of inbound and outbound traffic by running `iptables` or `nftables` commands.

3. **Ztunnel** (DaemonSet): Istio ambient mode component that provides the L4 data plane (mTLS, L4-level authorization) without adding a sidecar to the user's application. Runs on every cluster node.

   The component is created by the Deckhouse controller if the `.settings.ambient.enabled` module parameter is set to `true` (disabled by default).

   It consists of a single container:

   - **istio-proxy**: Main container that receives configuration via xDS from the istiod controller.

4. **Waypoint-controller** (Deployment): controller for the ambient mode L7 layer. The controller manages the WaypointInstance custom resource and creates or updates the corresponding `waypoint-<NAME>` Deployment.

   Created under the same conditions as ztunnel.

   It consists of a single container:

   - **waypoint-controller**: Main container.

5. **Waypoint-<NAME>** (Deployment): proxying service that handles L7 traffic (HTTP routing and authorization) for the services or workloads of the given namespace. Created dynamically by the waypoint-controller component for each WaypointInstance resource.

   It consists of a single container:

   - **istio-proxy**: Main container that receives configuration via xDS from the istiod controller.

Federation or multicluster mode components

> ⚠ Cross-cluster interaction (federation or multicluster) is only available between DP clusters, since DP installs a modified version of Istio that is not compatible with vanilla Istio in other clusters.

The following additional components are created when Istio runs in federation or multicluster mode:

1. **Metadata-exporter** (Deployment): component that provides public cluster metadata (the CA root certificate, public keys, endpoint addresses) to remote clusters for configuring cross-cluster interaction (federation or multicluster).

   The component is created by the Deckhouse controller if the `.settings.federation.enabled` or the `.settings.multicluster.enabled` module parameter is enabled (both disabled by default).

   It consists of the following containers:

   - **metadata-exporter**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to metrics.

2. **Alliance-healthcheck** (Deployment): component that checks the availability of the mesh connection with remote clusters and updates the status of the IstioFederation and IstioMulticluster custom resources.

   Created under the same conditions as metadata-exporter.

   It consists of a single container:

   - **healthcheck**: Main container.

3. **Ingressgateway** (DaemonSet): component that accepts mesh traffic from remote clusters via mTLS with SNI passthrough. A separate component from ingress-gateway-controller-<NAME>, dedicated exclusively to federation and multicluster traffic.

   The component is created by the Deckhouse controller if either:

   - The `.settings.federation.enabled` module parameter is enabled.
   - The `.settings.multicluster.enabled` module parameter and the `.spec.enableIngressGateway` parameter of the IstioMulticluster custom resource are both enabled.

   It consists of a single container:

   - **istio-proxy**: Main container that receives configuration via xDS from the istiod controller.

4. **Metrics-exporter** (Deployment): component that collects data for multicluster configuration metrics.

   The component is created by the Deckhouse controller if the `.settings.multicluster.enabled` module parameter is enabled.

   It consists of the following containers:

   - **metrics-exporter**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to metrics.

5. **Api-proxy** (Deployment): component that provides remote clusters with read access to this cluster's service mesh configuration and application resources. Used by the remote cluster's istiod for multicluster service discovery.

   The component is created by the Deckhouse controller if the `.settings.multicluster.enabled` module parameter is enabled.

   It consists of a single container:

   - **api-proxy**: Main container.

Component that is not part of the `istio` module:

- **User application**: workload created by the user and modified by istiod when the pod is mutated.

  It consists of the following containers:

  - **istio-init**: optional init container that sets up iptables rules to intercept application traffic. Added if the `.settings.dataPlane.trafficRedirectionSetupMode` module parameter is set to `InitContainer`. When it is set to `CNIPlugin`, this function is performed by the istio-cni-node component instead.
  - **istio-proxy**: sidecar container that lets the user application participate in the Istio mesh network. Added if the `.settings.ambient.enabled` module parameter is set to `false` (disabled by default).
  - **user-app**: the user application's own set of init and sidecar containers.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Authorizes requests for the module components' metrics.
   - Manages the Istio, IstioRevision, IngressIstioController, IstioFederation, IstioMulticluster, and WaypointInstance custom resources.
   - Manages custom resources of the `networking.istio.io`, `security.istio.io`, `telemetry.istio.io`, and `extensions.istio.io` API groups.
   - Creates and manages the `istiod-<VERSION>` and `waypoint-<NAME>` Deployments and the `ingress-`

    `gateway-controller-<NAME>` DaemonSet.
    - Reads Pod, Namespace, Node, Service, Secret, ConfigMap, Job, CronJob, Deployment, DaemonSet, ReplicaSet, and StatefulSet resources.
2. **The `user-authn` module**: Authenticates Kiali web interface users.
3. **Trickster**: Queries service mesh traffic metrics for the Kiali web interface.
4. **Remote DP cluster**:
    - Checks the availability of the mesh connection between clusters.
    - Retrieves the remote cluster's service mesh and application parameters.

The following external components interact with the module:

1. **Kube-apiserver**:
    - Validates custom resources of the `networking.istio.io`, `security.istio.io`, `telemetry.istio.io`, and `extensions.istio.io` API groups.
    - Mutates pods to add init and sidecar containers.
2. **Prometheus-main**: Collects metrics from all module components.
3. **Containerd**: Runs CNI plugin binaries.
4. **Load balancer**: Balances incoming traffic to ingress-gateway-controller.
5. **Gateway/Ingress controller**: Forwards the authenticated user request to the Kiali web interface. Depends on the chosen method of publishing resources: using the Ingress controller of the `ingress-nginx` module or the Gateway controller of the `alb` module.
6. **Remote DP cluster**:
    - Requests the cluster's public metadata.
    - Sends mesh traffic via mTLS SNI passthrough.
    - Reads the cluster's service mesh and application parameters.
7. **User application**: Receives configuration via xDS from the istiod controller.

Additional resources

- Module `alb` documentation
- Module `cni-cilium` documentation
- Module `deckhouse` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation
- Module `user-authn` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Metallb module

The `metallb` module implements the LoadBalancer mechanism for services in bare-metal clusters.

The following operating modes are supported:

- **Layer 2**: Implements an improved load balancing mechanism (compared to the standard L2 mode in MetalLB) for bare-metal clusters. It allows using multiple "public" IP addresses for services and distributing them evenly across available cluster nodes.
- **BGP**: Fully based on the MetalLB solution and uses the BGP protocol to create routes.

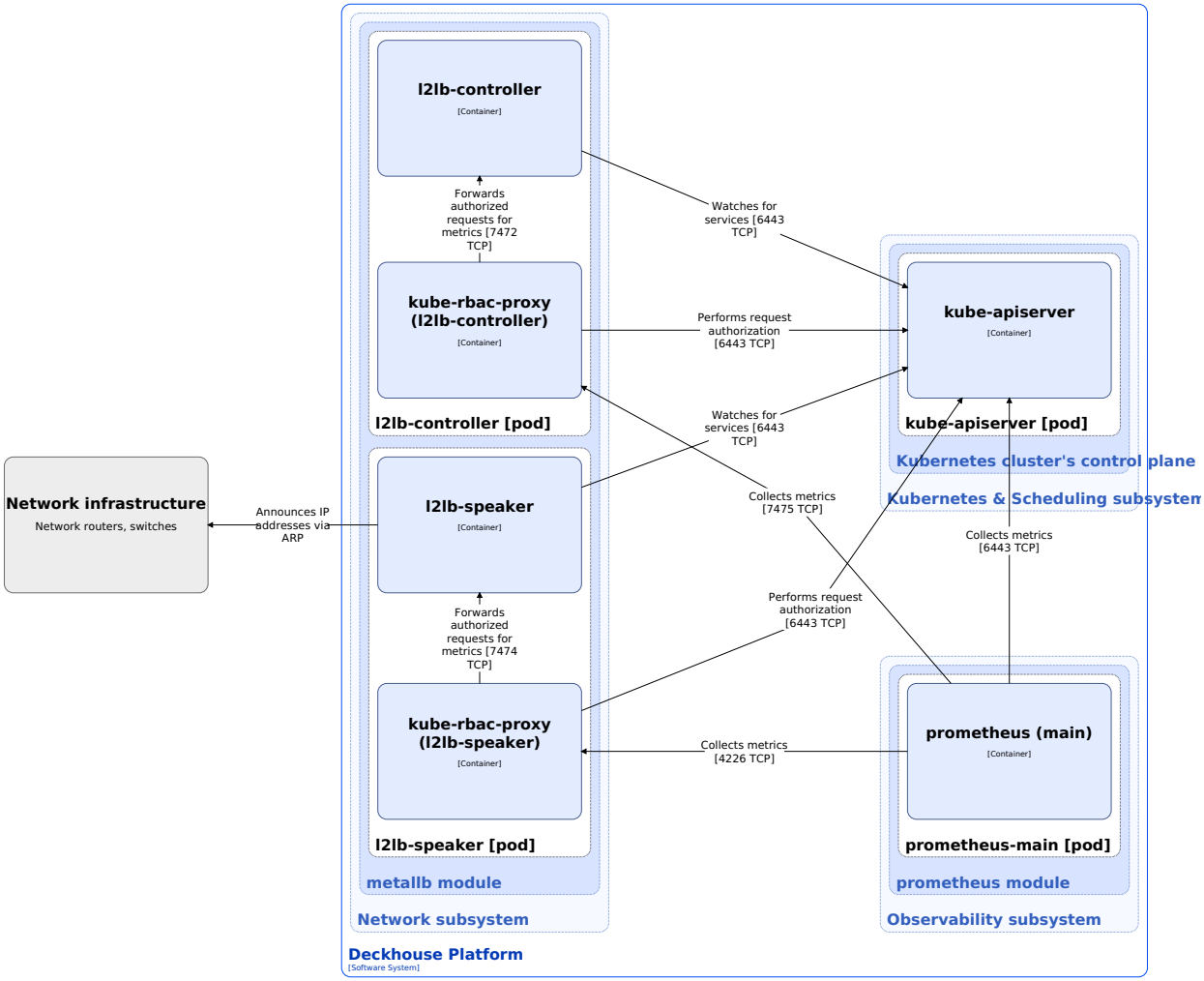
For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

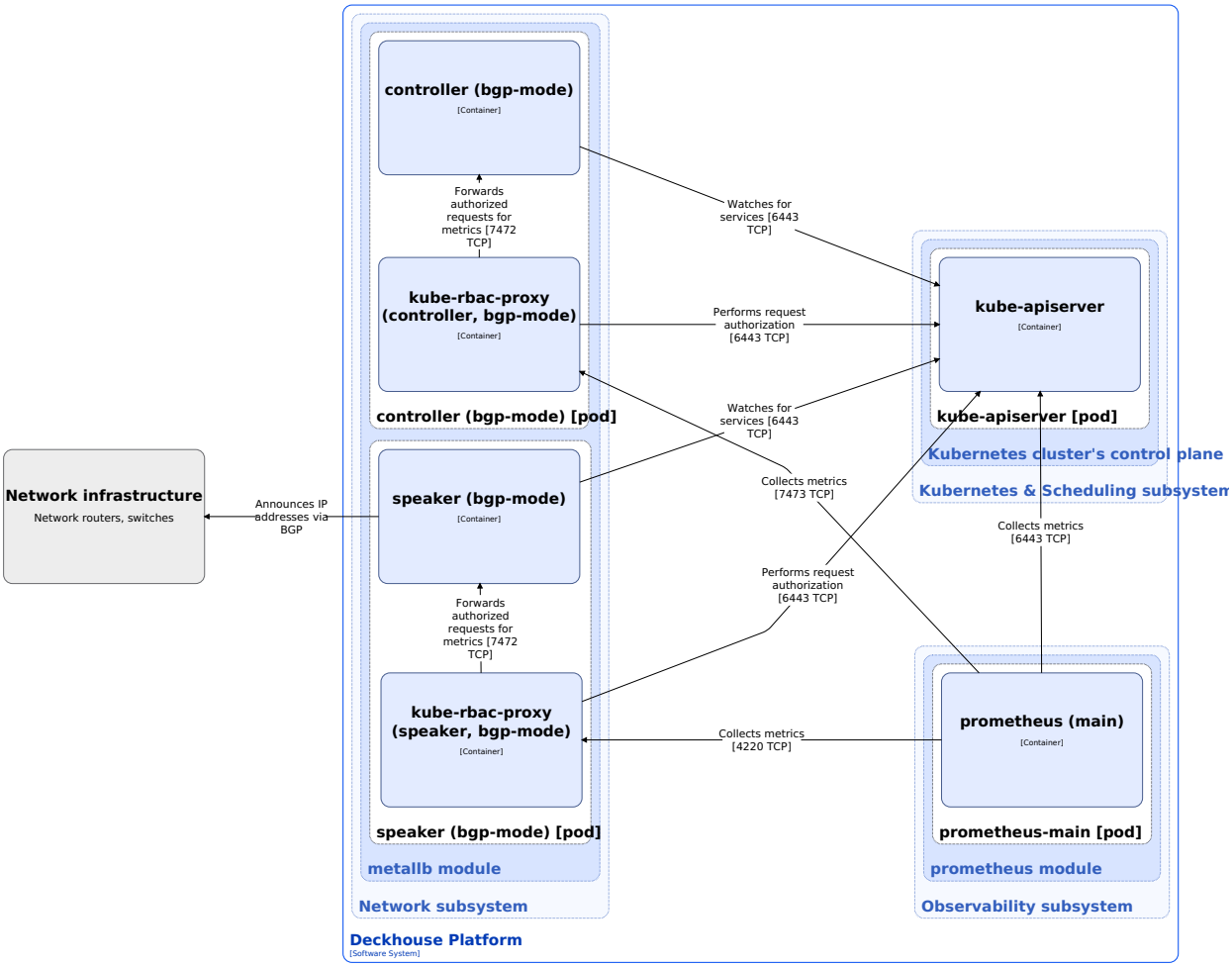
> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `metallb` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams.

MetalLB in Layer 2 mode:



MetalLB in BGP mode:



Module components

The module consists of the following components:

1. **Controller/l2lb-controller** (Deployment): MetalLB controller responsible for assigning IP addresses to LoadBalancer services.

   The controller monitors changes in Kubernetes services and applies IP address configuration based on a predefined address pool specified in the module configuration.

   It consists of the following containers:

   - **controller/l2lb-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to controller metrics. It is an open-source project.

2. **Speaker/l2lb-speaker** (DaemonSet): MetalLB speaker running on each cluster node that participates in load balancing. It is responsible for implementing the load balancing mechanism at the network level.

   Depending on the operating mode, it performs the following functions:

   - In **Layer 2 mode**, it uses ARP to associate virtual IP addresses of services with the physical addresses of cluster nodes.
   - In **BGP mode**, it announces routes to virtual IP addresses via the BGP protocol, making services accessible from outside the cluster.

   It consists of the following containers:

   - **speaker/l2lb-speaker**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to speaker metrics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Monitors changes in Kubernetes services and applies IP address configuration.
   - Authorizes requests for metrics.

2. **Network equipment**: Ensures the availability of virtual IP addresses outside the cluster:

   - In **BGP mode**, it advertises routes to virtual IP addresses using the BGP protocol.

    - In **Layer 2 mode**, it uses ARP/GARP to notify upstream routers that the virtual IP addresses are associated with the MAC addresses of specific nodes.

The following external components interact with the module:

- **Prometheus-main**: Collects metrics from the MetalLB controller and speaker.

Additional resources

- Module `metallb` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Sdn module

The `sdn` module provides software-defined networking (SDN) functions in Deckhouse Platform (DP):

- Configuring network interfaces on nodes.
- Additional networks for pods and virtual machines.
- IPAM (IP Address Management) for additional networks. The IPAM mechanism allocates IPv4 addresses from pools and assigns them to additional network interfaces of pods connected to cluster networks (ClusterNetwork) and project networks (Network).
- Underlay networks for passing through hardware network devices.
- System networks for node-level service traffic.

The module works with the following custom resources of the `network.deckhouse.io` API group:

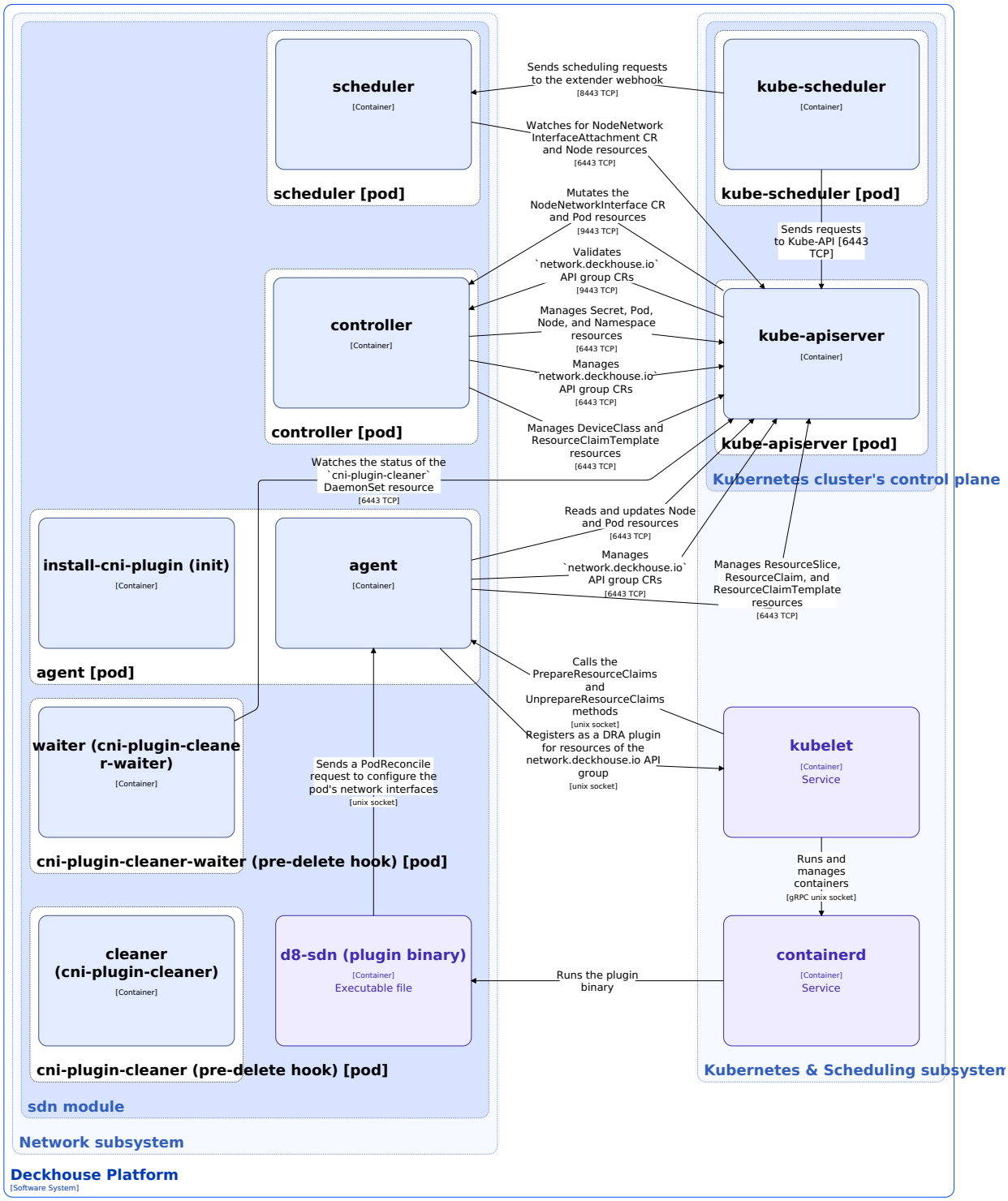
- ClusterIPAddress: Represents a request for an IPv4 address at the cluster scope, for a network interface connected to a ClusterNetwork or SystemNetwork.
- ClusterIPAddressPool: Defines a cluster-scoped pool of IPv4 addresses used for IPAM in cluster networks.
- ClusterNetwork: Defines a publicly available network at the cluster scope.
- IPAddress: Represents a request for an IPv4 address for an additional network interface of a pod connected to a project network.
- IPAddressLease: A reservation record for an allocated IP address that prevents it from being assigned again.
- IPAddressPool: Defines a pool of IPv4 addresses within a user namespace, used for IPAM in a project network.
- Network: A namespaced resource that creates a dedicated project network based on a network class provided by the administrator in a NetworkClass resource.
- NetworkClass: Defines a range of VLAN tags and node network interfaces that users can base their own Network resources on.
- NodeNetworkInterface: Describes a network interface on a node, used both to discover existing interfaces and to create new ones (VLAN, Bond).
- SystemNetwork: Creates an additional service network on cluster nodes on top of an existing underlay network for node-level service traffic.
- UnderlayNetwork: Provides direct pass-through of physical and virtual node network interfaces (PF/VF) to pods via Dynamic Resource Allocation (DRA), in Shared or Dedicated mode.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `sdn` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller** (Deployment): the controller manages software-defined networks in DP, performing the following actions:

   - Manages custom resources of the `network.deckhouse.io` API group.
   - Validates custom resources of the `network.deckhouse.io` API group using the Validating Admission Controllers mechanism.
   - Mutates NodeNetworkInterface custom resources using the Mutating Admission Controllers mechanism.
   - Mutates Pod resources in namespaces labeled `direct-nic-access.network.deckhouse.io/enabled` to inject a ResourceClaim for hardware network device pass-through.

   It consists of a single **controller** container.

2. **Agent** (DaemonSet): a component that runs on every cluster node and performs the following operations:

   - Configures node network interfaces (port bonding, bridging, VLAN interface configuration).
   - Brings up additional, underlay, and system networks.
   - Implements a CNI server that the d8-sdn component connects to over a Unix socket.
   - Implements a Dynamic Resource Allocation (DRA) driver for the `network.deckhouse.io` API group,

registering itself as a DRA plugin with kubelet, for passing physical and virtual network devices through to a pod. This functionality is available in the DP Ultimate/EE, Certified Core/CSE Lite (1.73), and Certified Pro/CSE Pro (1.73) editions.

The agent translates cluster node and pod configurations into low-level network interface and eBPF program settings that actually process traffic.

It consists of the following containers:

- **install-cni-plugin**: Init container that copies the `d8-sdn` CNI plugin executable to the `/opt/cni/bin` directory on the node.
- **agent**: Main container.

> ⚠ The agent component has privileged access to the operating system of each node. On Linux, this requires the following capabilities:
>
> - NET_ADMIN
> - SYS_ADMIN
> - BPF
>
> The container also runs as the root user (`runAsUser: 0`), uses `hostNetwork: true`, and the `Unconfined` AppArmor profile. This is required to configure node network interfaces, network namespaces, and to load BPF programs.

3. **D8-sdn**: A CNI plugin executable run by the containerd component according to the CNI specification, for example, ADD when creating a pod's network namespace and DEL when removing it. D8-sdn communicates with the agent's API over a Unix socket and triggers the low-level network configuration for pods.
4. **Scheduler** (Deployment): a kube-scheduler extender that suggests nodes for pod placement according to the pods' network interface settings.
5. **Cni-plugin-cleaner** (DaemonSet): a Helm `pre-delete` module hook, triggered by the Deckhouse controller before the `sdn` module is removed. It runs on every node and deletes the CNI plugin files (the executable in `/opt/cni/bin` and the configuration in `/etc/cni/net.d`) installed by the agent component. It does not take part in the module's normal operation.
6. **Cni-plugin-cleaner-waiter** (Job): a Helm `pre-delete` module hook, triggered by the Deckhouse controller before the `sdn` module is removed, that waits for the `cni-plugin-cleaner` DaemonSet to finish on every cluster node before Helm is allowed to complete removing the module release.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages custom resources of the `network.deckhouse.io` API group.
   - Manages Secret, Pod, Node, Namespace, DeviceClass, ResourceSlice, ResourceClaim, and ResourceClaimTemplate resources.
2. **Kubelet**: Registers itself as a DRA plugin for resources of the `network.deckhouse.io` API group.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates custom resources of the `network.deckhouse.io` API group.
   - Mutates NodeNetworkInterface and Pod resources.
2. **Kube-scheduler**: Sends requests to the `sdn-scheduler-extender` webhook for scheduling pods that use software-defined networks.
3. **Containerd**: Runs the d8-sdn executable file with certain commands according to the CNI specification, for example, ADD when creating a pod's network namespace and DEL when removing it.
4. **Kubelet**: Calls the `PrepareResourceClaims` and `UnprepareResourceClaims` RPC-methods to configure a pod's DRA resources.

Additional resources

- Module `sdn` documentation

> ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Service-with-healthchecks module

The `service-with-healthchecks` module provides an internal load balancer with per-port health checks.

The module works with the ServiceWithHealthchecks custom resource, which defines a Service load balancer with an

independent TCP, HTTP, or PostgreSQL health check for each port.

Unlike the regular load balancer, in which readiness probes are tied to the container state, ServiceWithHealthchecks custom resource allows you to set up active probes on individual TCP ports. In this way, each of the load balancers dealing with the same pod can operate independently of the others: a pod that fails the probes of one balancer is still published by the others.

Probes are grouped by the port they check, not by the resource. When a probe fails, the pod stops receiving traffic on that port only, and the other ports of the same resource keep working. When several probes check the same port, all of them have to pass for that port to receive traffic.

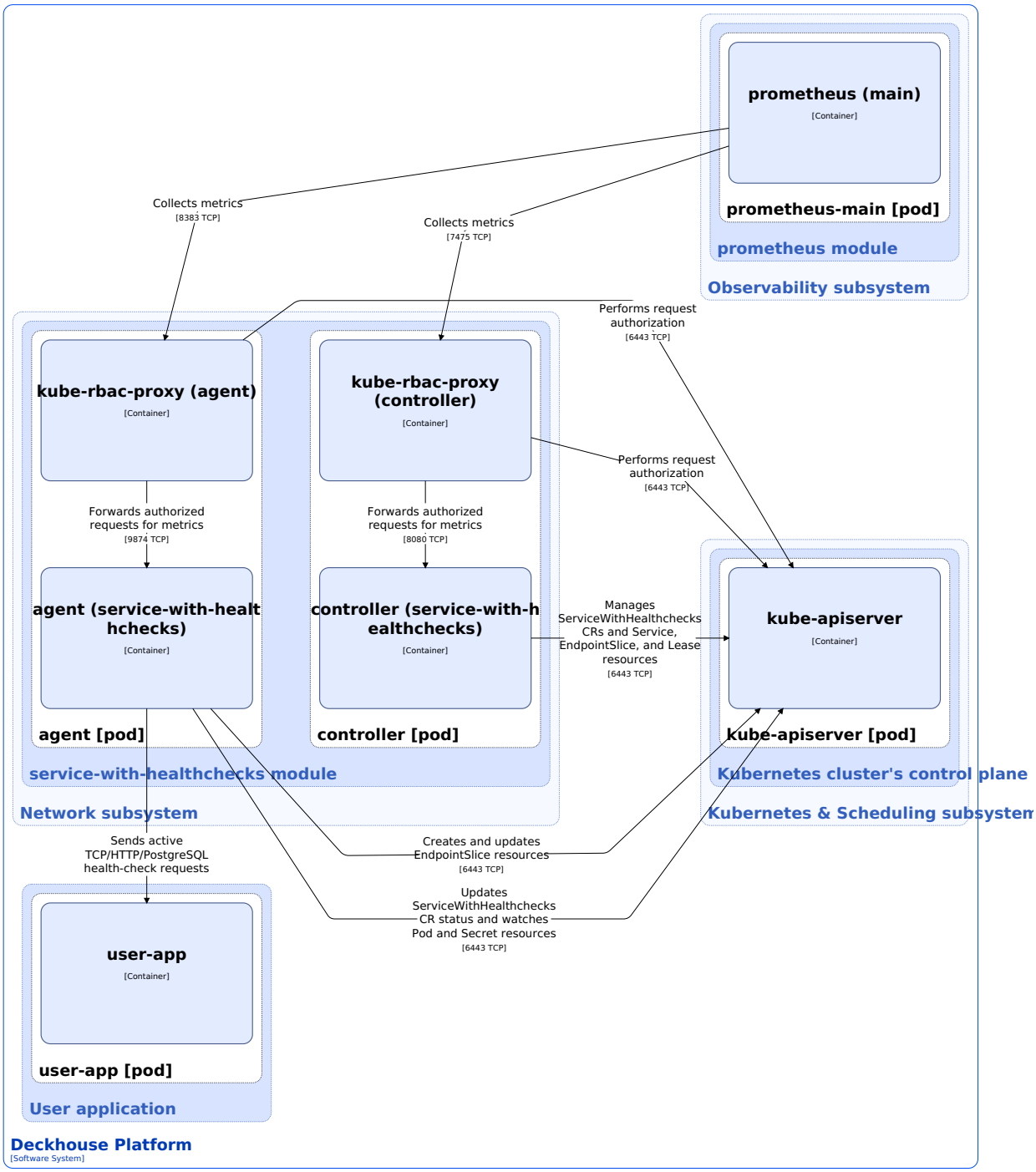
The module is CNI-agnostic: the controller and agent only manage standard Kubernetes Service and EndpointSlice objects, so traffic is still routed by the cluster's existing kube-proxy or CNI implementation, exactly as for a regular Service.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `service-with-healthchecks` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller** (Deployment): Reconciles ServiceWithHealthchecks resources and performs the following actions:

   - Creates and maintains a Service resource with an empty `selector` for each custom `ServiceWithHealthchecks` resource. Load balancing is determined entirely by the EndpointSlice resources created by the agent component for the corresponding Service resources.
   - Removes obsolete EnpointSlice resources after cluster node removal.

   It consists of the following containers:

   - **controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to controller metrics.

2. **Agent** (DaemonSet): Runs on every cluster node and performs active TCP, HTTP, or PostgreSQL health checks against the pods scheduled to its own node according to the specification of the corresponding ServiceWithHealthchecks custom resource. Based on the check results, agent creates and updates an EndpointSlice bound to the child Service maintained by the controller. Component also watches Secret resources of the `network.deckhouse.io/postgresql-credentials` type, used to authenticate PostgreSQL health checks.

   It consists of the following containers:

- **agent**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to component metrics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches and updates the status of ServiceWithHealthchecks custom resources.
   - Watches Pod resources on its own node and Secret resources of the `network.deckhouse.io/postgresql-credentials` type.
   - Manages Service, EndpointSlice, and Lease resources.
   - Authorizes requests for metrics.
2. **User services deployed in DP**: Sends active TCP, HTTP, or PostgreSQL health checks to the ports of pods on its node specified in the ServiceWithHealthchecks resource.

The following external components interact with the module:

1. **Prometheus-main**: Collects metrics from the agent and controller components.

Additional resources

- Module `service-with-healthchecks` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Static-routing-manager module

The `static-routing-manager` module manages static routes and `ip rule` entries on cluster nodes.

The module works with the following custom resources of the `network.deckhouse.io` API group:

- IPRuleSet: Describes a set of `ip rule` entries to be created on nodes matching a `nodeSelector`.
- RoutingTable: Describes a routing table and the routes it contains, for nodes matching a `nodeSelector`.

The module also works with the following internal custom resources of the `internal.network.deckhouse.io` API group:

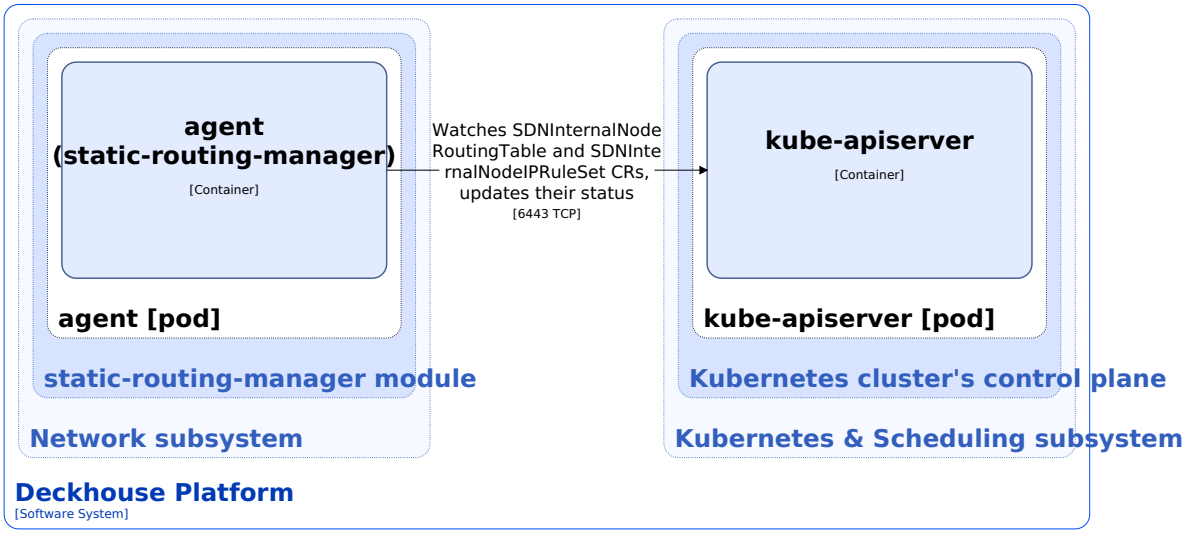
- SDNInternalNodeIPRuleSet: Contains the set of `ip rule` entries for a specific cluster node.
- SDNInternalNodeRoutingTable: Contains the routing table for a specific cluster node.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `static-routing-manager` module and its interactions with other components of Deckhouse Platform are shown in the following diagram:



Module components

The module consists of one **agent** (DaemonSet) component and one **agent** container. Agent runs on every cluster node and performs the following actions:

- Watches SDNInternalNodeRoutingTable and SDNInternalNodeIPRuleSet custom resources and updates their status.
- Applies the static routes and `ip rule` entries to the node's kernel routing table.

The SDNInternalNodeRoutingTable and SDNInternalNodeIPRuleSet custom resources are created by the Deckhouse controller of the `deckhouse` module for each cluster node matching the `nodeSelector` of the IPRuleSet and RoutingTable resources.

Module interactions

The module interacts with **kube-apiserver**:

- Manages the SDNInternalNodeRoutingTable and SDNInternalNodeIPRuleSet custom resources.
- Watches the IPRuleSet and RoutingTable custom resources and updates their status.

Additional resources

- Module `deckhouse` documentation
- Module `static-routing-manager` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

DNS modules

The `kube-dns` module provides domain name resolution based on CoreDNS in Deckhouse Platform (DP).

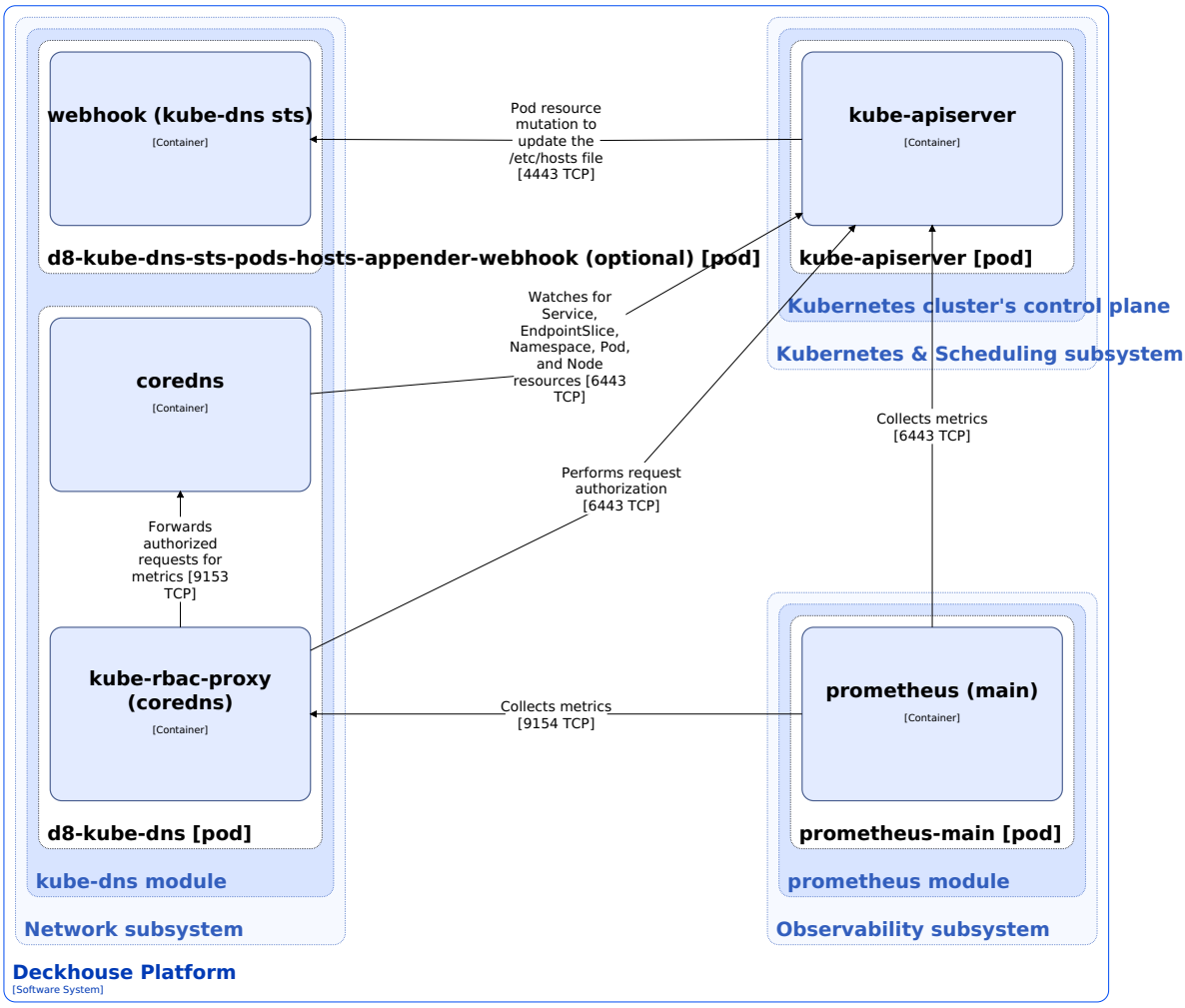
For more details about module configuration and usage examples, refer to the corresponding documentation section.

kube-dns module

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `kube-dns` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The `kube-dns` module consists of the following components:

1. **D8-kube-dns** (Deployment): Main module component implementing a DNS server in the Kubernetes cluster.

   The d8-kube-dns component watches changes in standard Service, EndpointSlice, Namespace, and Pod resources, and periodically requests Node resources. Based on the data it receives, d8-kube-dns updates records in the local object database.

   It consists of the following containers:

   - **coredns**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns metrics. It is an Open Source project.

2. **D8-kube-dns-sts-pods-hosts-appender-webhook** (Deployment): Optional component that consists of a single **webhook** container.

   The Deckhouse controller of the `deckhouse` module creates this component if the `.spec.settings.clusterDomainAliases` parameter is set in ModuleConfig.

   The component implements a mutating webhook server that adds the **render-etc-hosts-with-cluster-domain-aliases** init container to a pod created by a StatefulSet controller if the `.spec.subdomain` option is specified in the pod spec.

   The init container modifies the `/etc/hosts` file so that the name resolution subsystem works correctly with cluster domain aliases.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Watches standard Service, Endpoint, EndpointSlice, Namespace, Pod, and Node resources.
  - Authorizes requests for metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Modifies Pod resources created by the StatefulSet controller.
2. **Prometheus-main**: Collects module metrics.

node-local-dns module

The `node-local-dns` module provides a caching DNS service and a DNS request forwarding mechanism on each cluster node, reducing the load on `coredns` of the `kube-dns` module.

Depending on the CNI plugin used, the `node-local-dns` module implements different DNS request forwarding mechanisms:

- When using `cni-cilium`, Cilium handles DNS request forwarding using the CiliumLocalRedirectPolicy custom resource.
- When using `cni-flannel` or `cni-simple-bridge`, iptables handles DNS request forwarding.

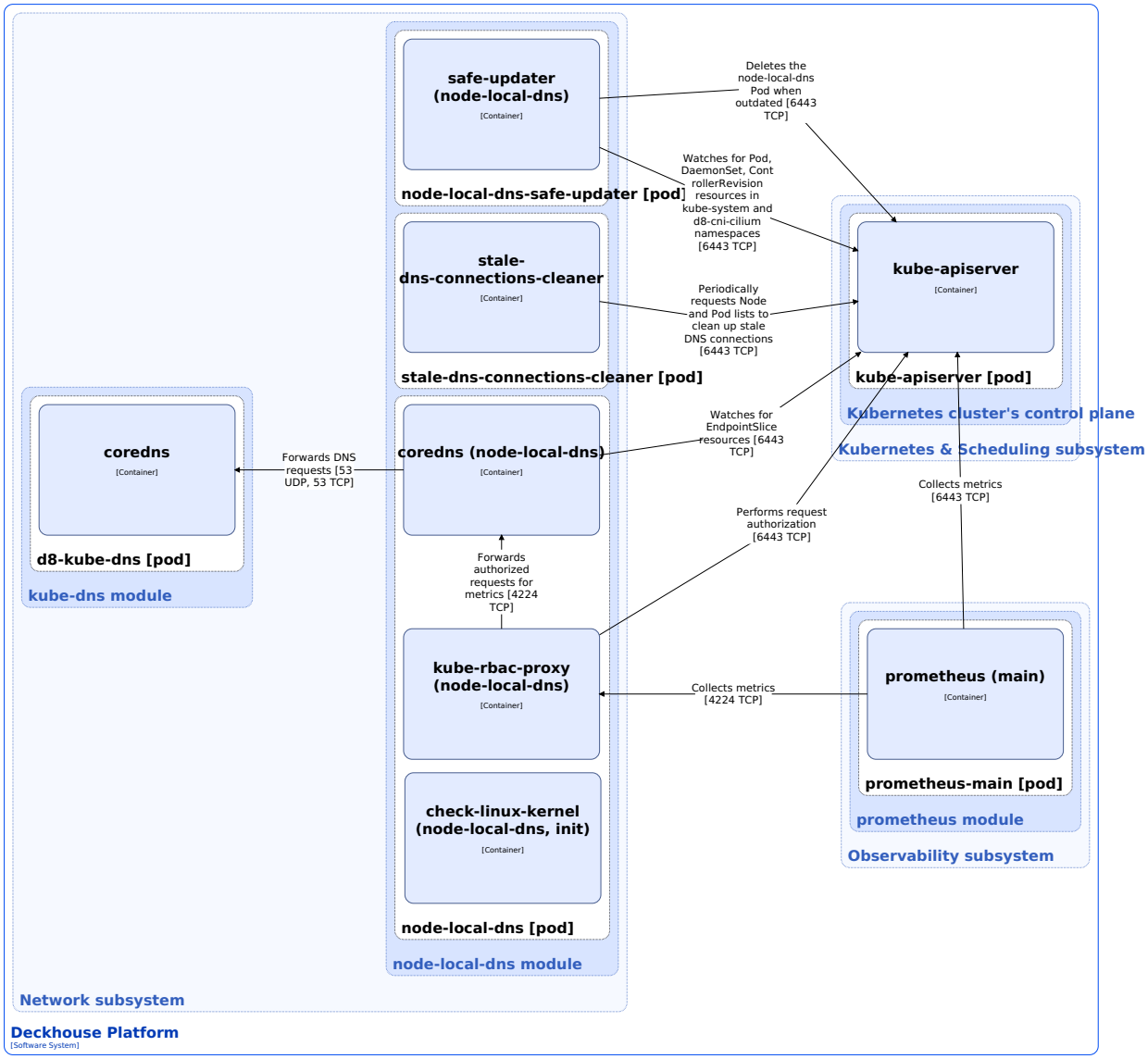
For more details about module configuration and usage examples, refer to the corresponding documentation section.

When using cni-cilium

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-local-dns` module when using Cilium as the CNI plugin and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The `node-local-dns` module consists of the following components:

1. **Node-local-dns** (DaemonSet): Main module component implementing a caching DNS server in the Kubernetes cluster.

   The node-local-dns component watches changes in standard EndpointSlice resources and uses them to update the list of DNS servers for forwarding requests.

   It consists of the following containers:

   - **check-linux-kernel**: Init container that checks the Linux kernel version.
   - **coredns**: Main container.
   - **cube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns metrics. It is an Open Source project.

2. **Stale-dns-connections-cleaner** (DaemonSet): Component that removes stale UDP connections left after the `node-local-dns` Pod is restarted. It consists of a single **stale-dns-connections-cleaner** container.

   > ⚠ The component has privileged access to the network subsystem of each node. On Linux, this requires the `CAP_NET_ADMIN` capability. This access is required to perform network connection operations at the Linux kernel level.

3. **Safe-updater** (Deployment): Component that provides safe restarts of node-local-dns when the DaemonSet spec changes.

   Safe-updater checks that Cilium is running on the node and is in a correct state, and only then sends a command to delete the `node-local-dns` Pod.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches standard EndpointSlice, DaemonSet, ControllerRevision, and Pod resources.
   - Periodically retrieves Node resources.
   - Deletes the `node-local-dns` Pod when the DaemonSet configuration becomes outdated.
   - Authorizes requests for metrics.

2. **D8-kube-dns**: Executes DNS requests.

The following external components interact with the module:

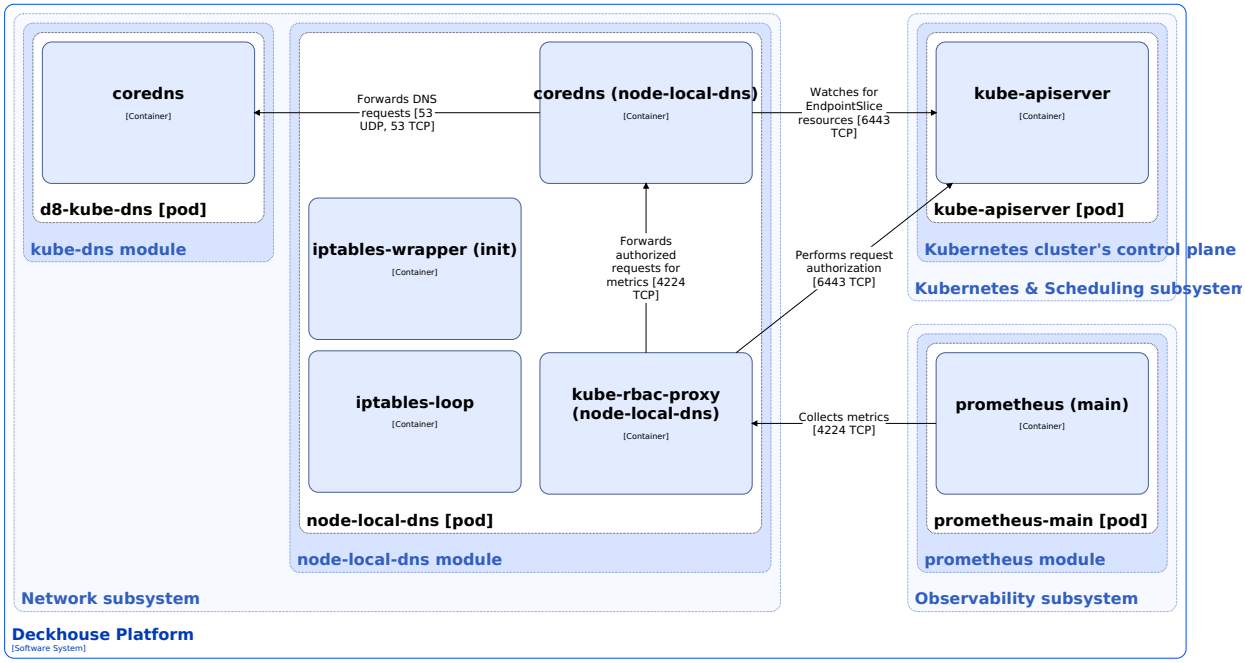
- **Prometheus-main**: Collects module metrics.

When using cni-flannel or cni-simple-bridge

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-local-dns` module when using the `cni-flannel` or `cni-simple-bridge` CNI plugin and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The `node-local-dns` module consists of the following components:

- **Node-local-dns** (DaemonSet): Main module component implementing a caching DNS server in the Kubernetes cluster.

  The node-local-dns component watches changes in standard EndpointSlice resources and uses them to update the list of DNS servers for forwarding requests.

  It consists of the following containers:

  - **iptables-wrapper**: Init container that prepares executables required for working with iptables.
  - **coredns**: Main container.
  - **iptables-loop**: Sidecar container that updates iptables rules for DNS request forwarding based on node-local-dns readiness.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns container metrics. It is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches standard EndpointSlice resources.
   - Authorizes requests for metrics.
2. **D8-kube-dns**: Executes DNS requests.

The following external components interact with the module:

- **Prometheus-main**: Collects module metrics.

Additional resources

- Module `kube-dns` parameters
- Module `node-local-dns` parameters
- Caching DNS server in a cluster
- Module `deckhouse` documentation
- Module `kube-dns` documentation

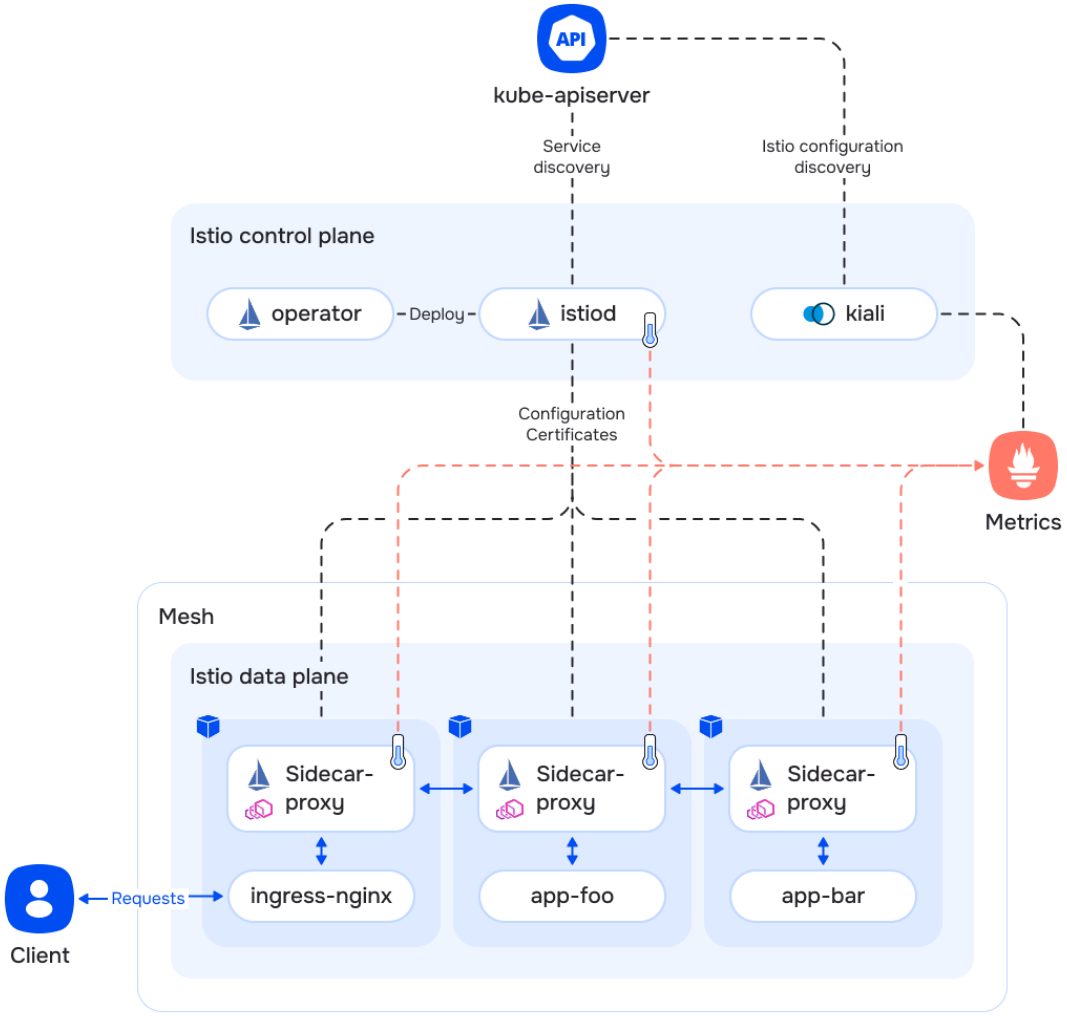
> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cluster-level Istio architecture

> **i** Compatibility between Istio versions and Kubernetes versions is defined by the compatibility table in the `istio` module documentation. Before using Istio, refer to the current versions and their status in that section.

The Istio components are divided into two categories:

- **Control plane**: Managing and maintaining services. The control plane usually refers to istiod Pods.
- **Data plane**: Istio application part composed of a set of sidecar-proxy containers.



All data plane services are grouped into a service mesh with the following features:

- It has a common namespace for generating service ID in the following format: `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. Each service mesh has a TrustDomain ID (in this case, it matches the cluster domain), for example, `mycluster.local/ns/myns/sa/myapp`.
- Authentication of services within a single service mesh using trusted root certificates.

Control plane components:

- **Istiod**: Main service working through the following tasks:
  - Continuous connection to the Kubernetes API and collecting information about services.
  - Processing and validating all Istio-related custom resources using the Kubernetes Validating Webhook mechanism.
  - Configuring each sidecar proxy individually:
    - Generating authorization, routing, balancing rules, etc.
    - Distributing information about other application services in the cluster.
    - Issuing individual client certificates for implementing Mutual TLS. These certificates are unrelated to the certificates that Kubernetes uses for its own service needs.
  - Automatic tuning of manifests that describe application Pods via the Kubernetes Mutating Webhook mechanism:
    - Injecting an additional sidecar-proxy service container.
    - Injecting an additional init container for configuring the network subsystem (configuring DNAT to intercept application traffic).
    - Routing readiness and liveness probes through the sidecar-proxy.
- **Operator**: Installs all the resources required to operate a specific version of the control plane.
- **Kiali**: Dashboard for monitoring and controlling Istio resources as well as user services managed by Istio:
  - Visualizing relationships between services.
  - Diagnosing problematic relationships.

    - Diagnosing control plane health.

With Istio enabled, the Ingress controller behavior is changed as follows:

- A sidecar proxy is added to the controller Pods. It only handles traffic from the controller to the application services (the `enableIstioSidecar` parameter of the IngressNginxController resource).
- Services not managed by Istio continue to function as before, requests to them are not intercepted by the controller sidecar.
- Requests to services running under Istio are intercepted by the sidecar and processed according to Istio rules. For details on enabling Istio for applications, refer to the corresponding documentation page.

The istiod controller and sidecar-proxy containers export their own metrics that the cluster-wide Prometheus collects.

Estimating overhead

Using Istio will incur additional resource costs for both control plane (istiod controller) and data plane (istio sidecars).

Control plane

The istiod controller continuously monitors the cluster configuration, compiles the settings for the data plane istio sidecars and distributes them over the network. Therefore, the more applications and their instances, the more services, and the more frequently this configuration changes, the more computational resources are required and the greater the load on the network.

Two approaches are supported to reduce the load on controller instances:

- Horizontal scaling (the `controlPlane.replicasManagement` parameter of the `istio` module). The more controller instances, the fewer instances of istio sidecars to serve for each controller and the less CPU and network load.
- Data plane segmentation using the Sidecar resource (recommended approach). The smaller the scope of an individual istio sidecar, the less data in the data plane needs to be updated and the less CPU and network load.

A rough estimate of overhead for a control plane instance that serves 1000 services and 2000 istio sidecars is 1 vCPU and 1.5 GB RAM.

Data plane

The consumption of data plane resources (istio sidecars) is affected by many factors:

- Number of connections
- The intensity of requests
- Size of requests and responses
- Protocol (HTTP/TCP)
- Number of CPU cores
- Complexity of the service mesh configuration

A rough estimate of the overhead for an istio sidecar instance is 0.5 vCPU for 1000 requests/sec and 50 MB RAM.

Istio sidecars also increase latency in network requests, about 2.5 ms per request.

Additional resources

- Module `ingress-nginx` custom resources
- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Application service architecture with Istio enabled

> Compatibility between Istio versions and Kubernetes versions is defined by the compatibility table in the `istio` module documentation. Before using Istio, refer to the current versions and their status in that section.

Architecture features

- **Sidecar proxy**: Each application service Pod gets a sidecar container, a sidecar proxy, with two applications:
    - **Envoy**: Proxies application traffic. It is responsible for implementing all the Istio functionality, including routing, authentication, authorization, etc.
    - **Pilot-agent**: A part of Istio that is responsible for keeping the Envoy configuration up to date and has a built-in caching DNS server.
- **DNAT settings**:
    - DNAT of incoming and outgoing application requests in the sidecar proxy is configured in each Pod using an additional init container. Thus, traffic will be intercepted transparently for applications.
    - Since incoming traffic is redirected to the sidecar proxy, this also applies to readiness/liveness probes. Since the Kubernetes subsystem does not support Mutual TLS probes, all existing probes are redirected to a port in the sidecar proxy, which passes them to the application unchanged.
- **Ingress Controller**:
    - Each Pod of an Ingress controller also includes a sidecar proxy, which handles traffic between the controller

and services.
    - Incoming traffic from users is handled directly by the controller.
- **Ingress resources**: These resources require minimal modification in the form of adding annotations:
    - `nginx.ingress.kubernetes.io/service-upstream: "true"`: The Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is handled by the sidecar proxy. Use this option only if your service has a ClusterIP.
    - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"`: The Ingress controller's sidecar proxy makes routing decisions based on the `Host` header. If this annotation is omitted, the controller will leave a header with the site address (for example, `Host: example.com`).
- **Services**:
    - Service resources do not require modifications and continue to work without adaptation. Applications can still access service addresses like `servicename`, `servicename.myns.svc`, etc.
- **DNS queries**:
    - Internal DNS queries of Pods are transparently redirected to the sidecar proxy for processing to resolve DNS names of services from neighboring clusters.

User request lifecycle

Application with Istio disabled

Application with Istio enabled

Additional resources

- Module `istio` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Caching DNS server in a cluster

Standard DNS operation in Kubernetes comes with a series of issues that may cause degradation of key service performance indicators:

- By default, the Linux kernel does not cache DNS requests, so there is no local cache in Pods.
- All DNS requests from a container result in a network request to the cluster DNS. This is also true for requests to resources located on the same node.
- A Pod's DNS request is first resolved through cluster DNS zones, and only afterward sent to external DNS servers. For example, a request to `ya.com` will first be resolved through cluster zones like `cluster.local`, `svc.cluster.local`, and `<namespace>.svc.cluster.local`. Only after receiving negative responses (meaning, on the second attempt or later) it will be resolved correctly.

Any minor network delays can significantly degrade service quality due to the above-mentioned issues.

One possible solution is to install a DNS server on each node. In Deckhouse Platform, this is implemented using the `node-local-dns` module.

When using a caching DNS server, external requests (that are not already cached) will still be attempted to resolve through the internal zone chain first. Under high load (for example, with many repeated requests per second for the same records, which is common), caching is enough to significantly improve DNS resolution performance.

Caching DNS server operation principles

When the caching DNS server is deployed, the `node-local-dns` module applies the following configuration steps on each cluster node:

- Configuring an interface with the IP address of the `kube-dns` service's clusterIP.
- Starting a caching CoreDNS that listens on that address.
- Adding an iptables rule: if the socket is open, traffic is redirected to it. Otherwise, a standard Kubernetes routing via ClusterIP is used:

```
-A PREROUTING -d <kube-dns IP address> -m socket --nowildcard -j NOTRACK
```

CoreDNS configuration aspects

Key characteristics of the CoreDNS configuration:

- All requests are cached.
- All DNS requests are forwarded to the cluster DNS ClusterIP.

Grafana dashboard

The `Kubernetes / DNS (node local)` dashboard displays:

- General charts (providing an overall view of DNS performance),
- Per-node charts (helping investigate node-specific issues identified in the general charts),

- Upstream charts (helping evaluate performance of the cluster DNS and node DNS servers specified in `/etc/resolv.conf`).

Additional resources

- Module `node-local-dns` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Storage subsystem

This subsection describes the architecture of the Storage subsystem of Deckhouse Platform (DP).

The Storage subsystem includes the following modules:

- `local-path-provisioner`: Provides the local storage on Kubernetes nodes using `HostPath` volumes and creates StorageClass resources to manage the allocation of local storage.
- `snapshot-controller`: Enables snapshot support for compatible CSI-drivers in the Kubernetes cluster.
- `sds-local-volume`: Manages the local block storage based on LVM and enables creating StorageClass resources in Kubernetes using the LocalStorageClass resource.
- `sds-node-configurator`: Manages block devices and LVM on Kubernetes cluster nodes through Kubernetes custom resources.
- `sds-replicated-volume`: Manages replicated block storage based on DRBD. LINSTOR is used as a control plane.
- `storage-volume-data-manager`: Provides secure export and import of persistent volume contents over HTTP protocol.
- Modules that provide a CSI driver implementation for integration with various types of storage (software and hardware):
  - `csi-ceph`
  - `csi-hpe`
  - `csi-huawei`
  - `csi-netapp`
  - `csi-nfs`
  - `csi-s3`
  - `csi-scsi-generic`
  - `csi-vsphere`
  - `csi-yadro-tatlin-unified`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Local-path-provisioner module

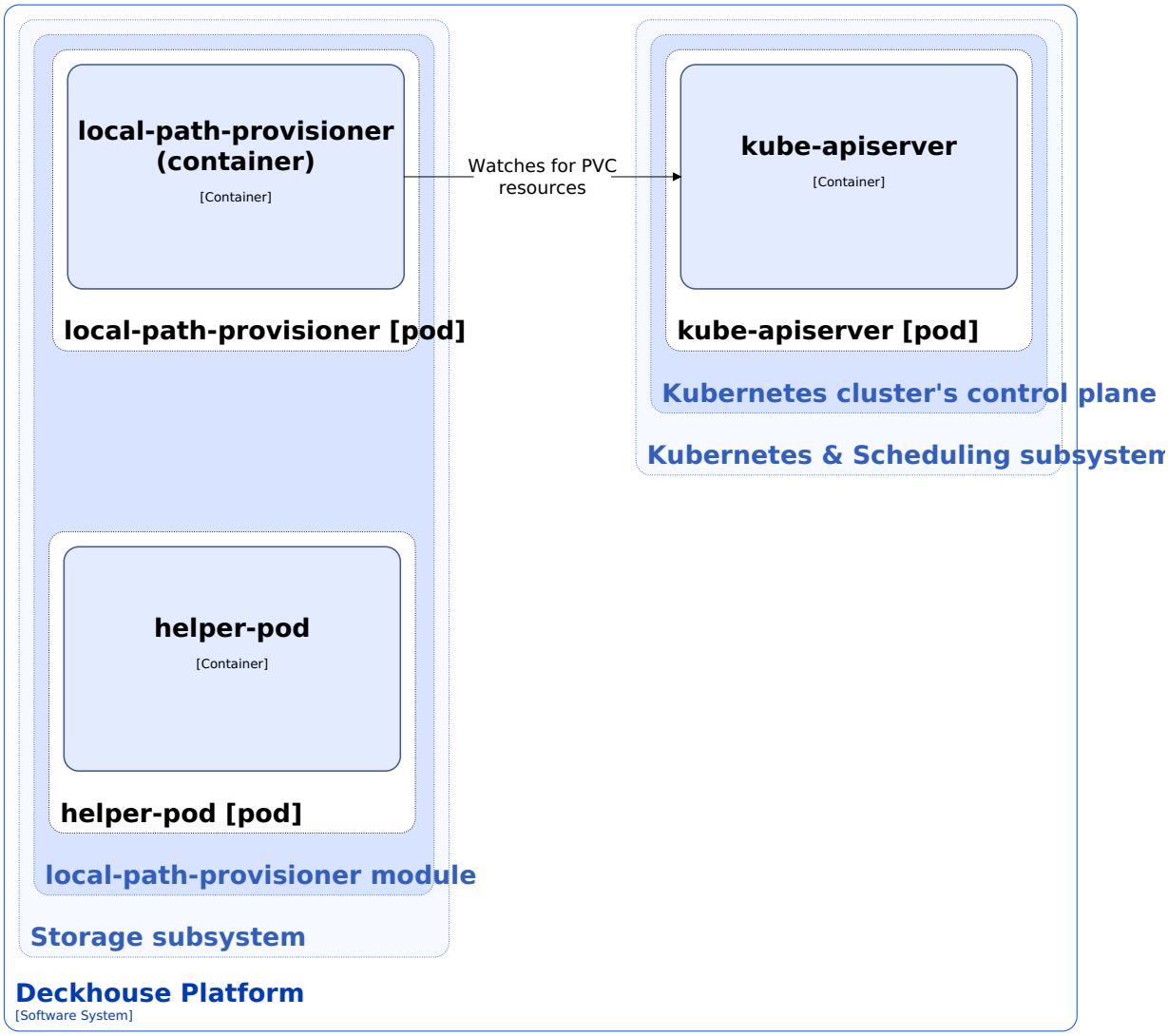
The `local-path-provisioner` module provides the local storage on Kubernetes nodes using `HostPath` volumes and creates StorageClass resources to manage the allocation of local storage.

For more details about module, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `local-path-provisioner` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Local-path-provisioner**: It performs the following actions when a pod orders a disk:
   - Creates a PersistentVolume of the `HostPath` volume type.
   - Creates a local disk directory for the volume on the target node. The directory path is based on the `path` parameter of the LocalPathProvisioner custom resource, as well as the PersistentVolume and PersistentVolumeClaim names.

   It consists of a single container:

   - **local-path-provisioner**: It is an open source project.
2. **Helper-pod**: It runs an installation script (`SETUP`) on the node before creating the volume to prepare the volume directory on the node and a cleanup script (`TEARDOWN`) after deleting the volume to clear the volume directory on the node.

   It consists of a single container:

   - **helper-pod**

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages PersistentVolumeClaim resources.

Additional resources

- Module `local-path-provisioner` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Snapshot-controller module

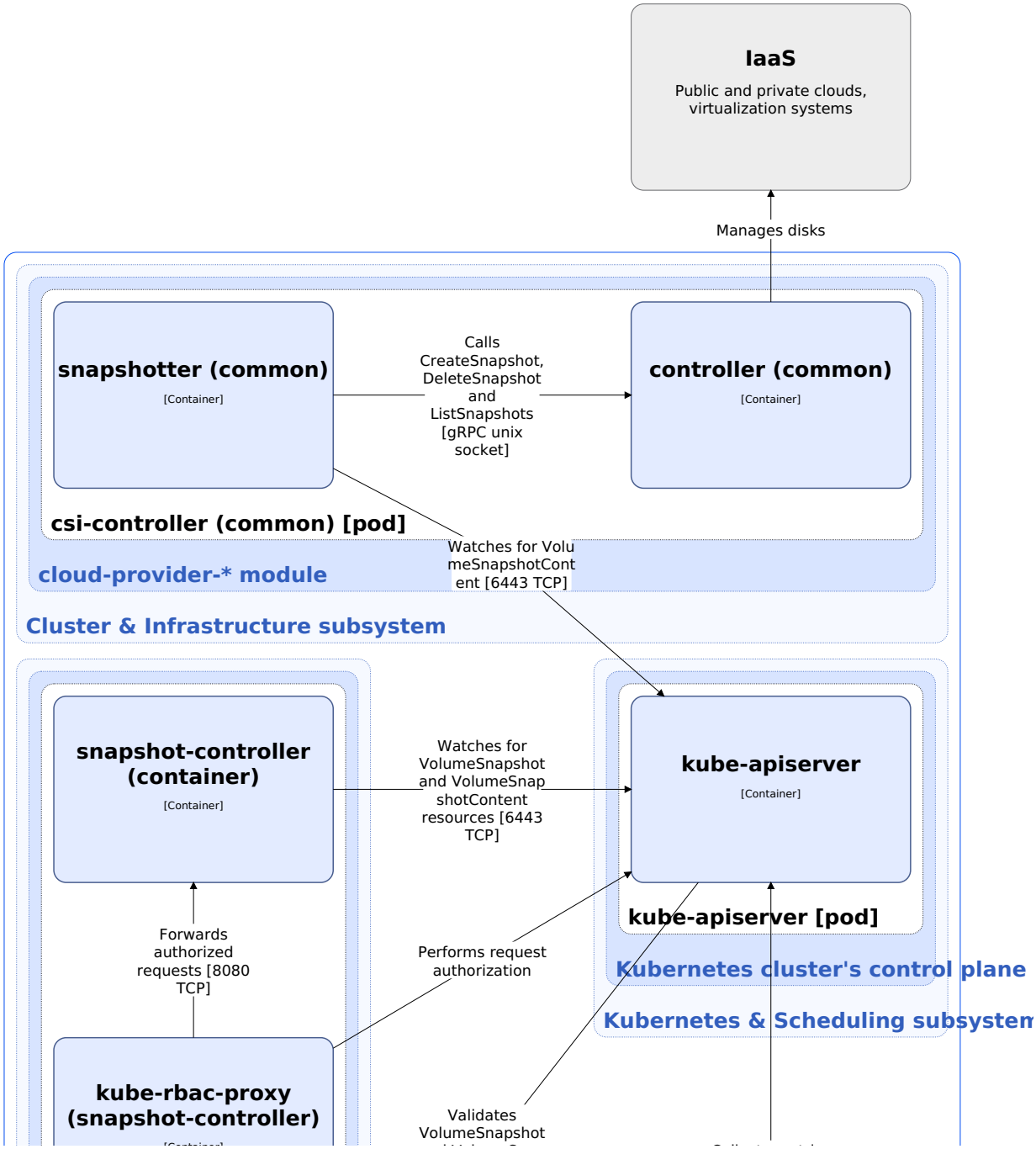
The `snapshot-controller` enables snapshot support for compatible CSI-drivers in Deckhouse Platform (DP).

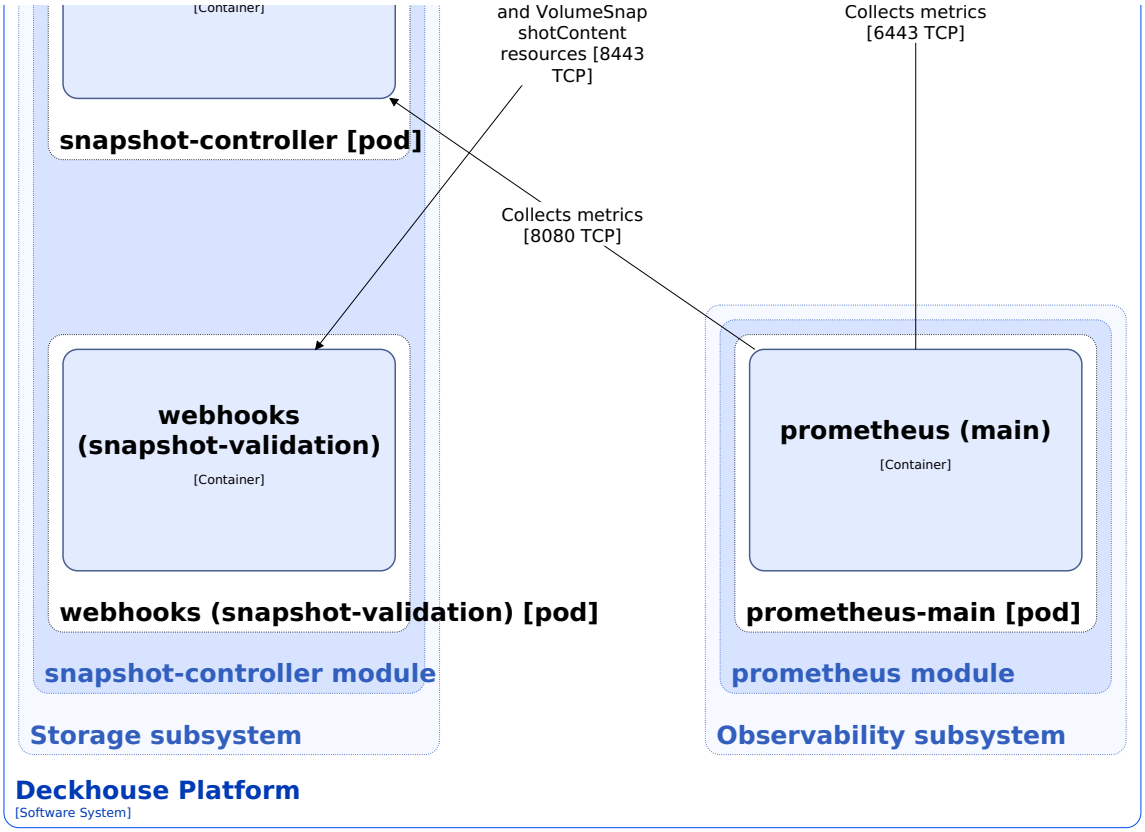
For more details about the module, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `snapshot-controller` module and its interactions with other components of DP are shown in the following diagrams:





Module components

The module consists of the following components:

1. **Snapshot-controller**: Works in conjunction with the snapshotter (external-snapshotter) sidecar container of the csi-controller Pod of the `cloud-provider-*` module (as long as the provider's CSI driver supports snapshot creation).

   A single snapshot-controller is used for all installed CSI drivers. It watches for VolumeSnapshot and VolumeSnapshotContent resources. When a new VolumeSnapshot resource is created, the controller creates a VolumeSnapshotContent resource and binds them together. As a result, VolumeSnapshot points to a respective VolumeSnapshotContent, and that VolumeSnapshotContent points to the original VolumeSnapshot.

   Snapshot creation is multi-step process:

   1. Snapshot-controller creates a VolumeSnapshotContent resource.
   2. The snapshotter sidecar triggers snapshot creation through csi-controller on the corresponding node and updates the VolumeSnapshotContent status (`snapshotHandle`, `creationTime`, `restoreSize`, `readyToUse`, and `error` fields).
   3. Snapshot-controller watches for the VolumeSnapshotContent status and keeps updating the VolumeSnapshot status until the bi-directional binding is complete and the `readyToUse` field is set to `true`. If any error occurs, the `error` field is updated accordingly.

   It consists of the following containers:

   - **snapshot-controller**: It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the controller metrics. It is an open-source project.

2. **Webhooks**: Component implementing a webhook server used for validating the VolumeSnapshot and VolumeSnapshotContent resources through Validating Admission Controllers. It consists of a single container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for VolumeSnapshot and VolumeSnapshotContent resources.
   - Authorizes requests for controller metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Uses the validation webhook to verify the created VolumeSnapshot and VolumeSnapshotContent resources.
2. **Prometheus-main**: Collects metrics from the controller.

Additional resources

- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Sds-local-volume module

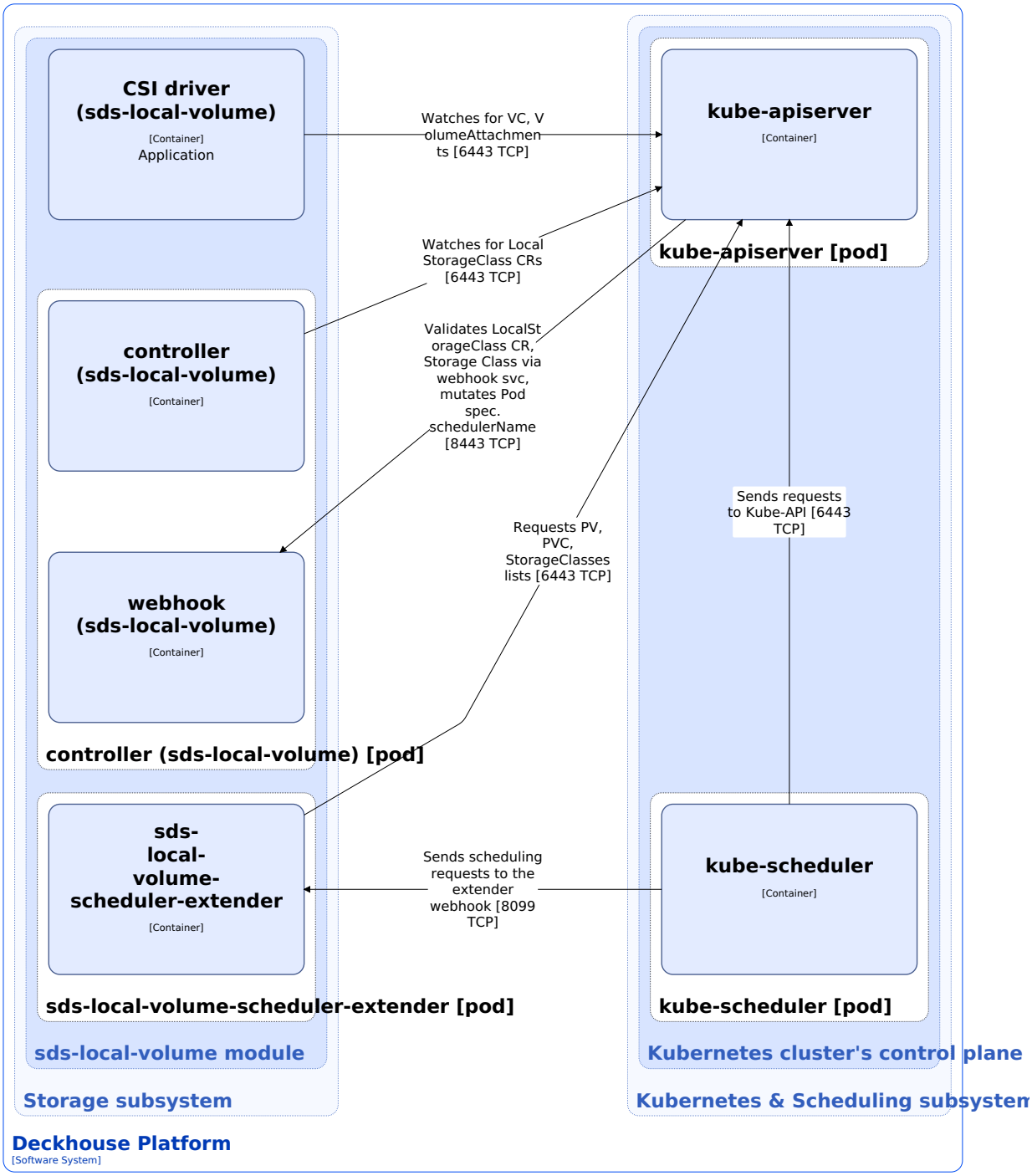
The `sds-local-volume` module is designed to manage local block storage based on LVM. It enables creating StorageClasses in Kubernetes using the LocalStorageClass resource.

For more details about module, refer to the module documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `sds-local-volume` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Controller**: It reconciles LocalStorageClass custom resources. LocalStorageClass is a Kubernetes custom resource that defines the configuration for Kubernetes StorageClass. The StorageClass being created uses `local.csi.storage.deckhouse.io` provisioner. StorageClass configures LVM logical volume types, VolumeGroups settings, reclaim policy, volume binding mode, etc. These settings are used by the provisioner of CSI driver (`sds-local-volume`) when managing LVM-based local volumes.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: A sidecar container that implements a webhook server for validating LocalStorageClass custom resources, StorageClass resources, as well as for mutating the `spec.schedulerName` attribute of pods using volumes created by `local.csi.storage.deckhouse.io` provisioner. As a result of the mutation, `sds-local-volume` is filled in the name of the scheduler (`spec.schedulerName`) in the pod specification, so that the placement of the pod is determined not by the standard Kubernetes scheduler (kube-scheduler), but by the `sds-local-volume-scheduler-extender` component from this module.

2. **Sds-local-volume-scheduler-extender**: It consists of a single container. It is a kube-scheduler extender, which implements a scheduling logic specific for pods using local volumes. When planning, the free space on the nodes used to place local volumes on them is taken into account, as well as the size of the disk space that needs to be reserved for these volumes.

3. **CSI driver (`sds-local-volume`)**: It is an implementation of the CSI driver for `local.csi.storage.deckhouse.io`. To study the CSI driver typical architecture used in DP, refer to the CSI-driver architecture documentation section. CSI driver (`sds-local-volume`) is developed by Flant.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolume, PersistentVolumeClaim, VolumeAttachment and StorageClass resources.
   - Reconciles LocalStorageClass custom resources.
   - Creates StorageClass resources.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates LocalStorageClass custom resources and StorageClass resources.
   - Mutates `spec.schedulerName` attribute for pods which use volumes created by `local.csi.storage.deckhouse.io` provisioner.
2. **Kube-scheduler**: Sends scheduling requests to the `sds-local-volume-scheduler-extender` webhook for the pods with `sds-local-volume` value specified in the `spec.schedulerName` attribute.

Additional resources

- Module `sds-local-volume` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Sds-node-configurator module

The `sds-node-configurator` manages block devices and LVM on Kubernetes cluster nodes through Kubernetes custom resources, performing following operations:

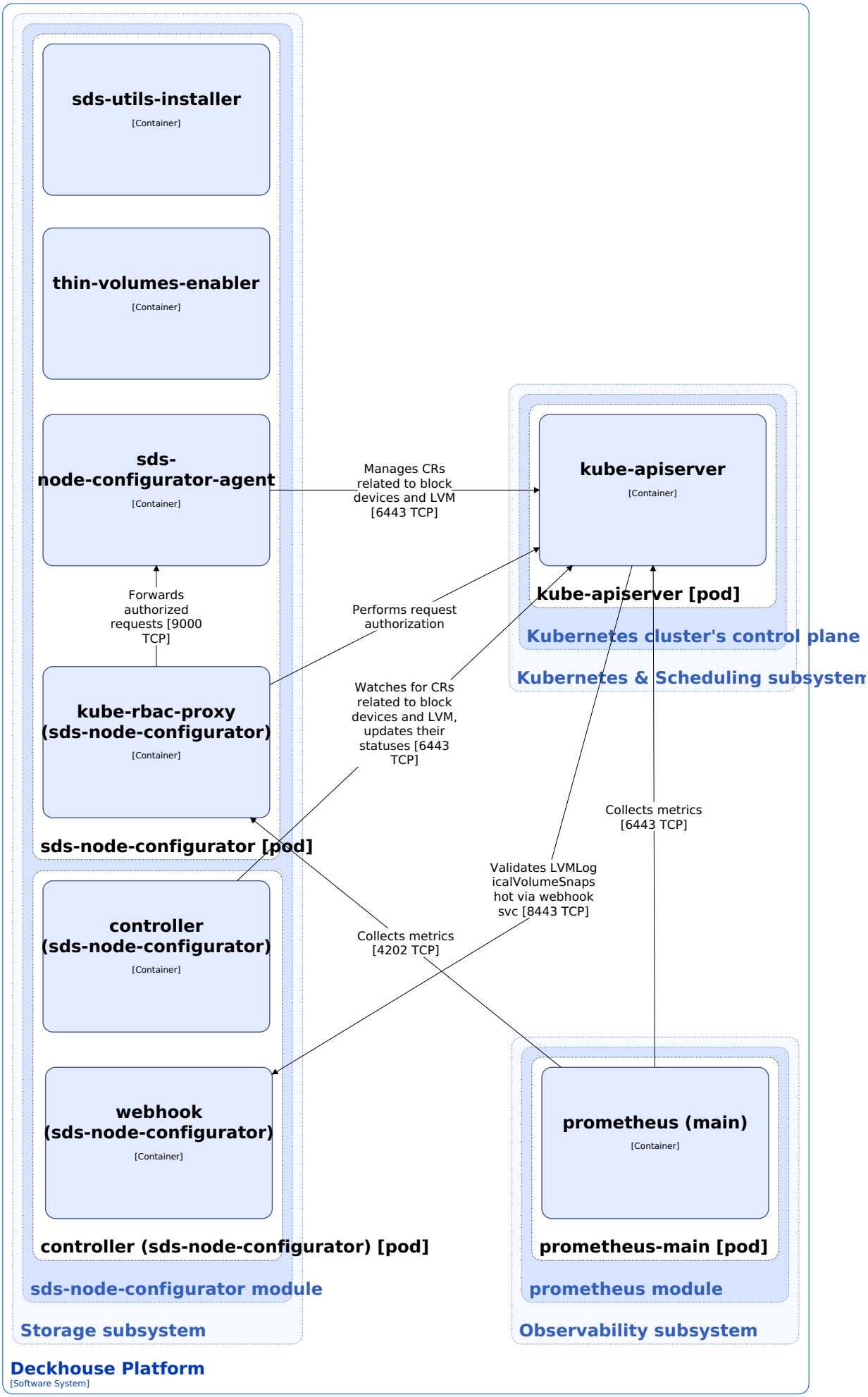
- Automatic discovery of block devices and creation/update/deletion of corresponding BlockDevice custom resources.
- Automatic discovery of LVM Volume Groups with `storage.deckhouse.io/enabled=true` LVM tag and thin pools on them on nodes, as well as management of corresponding LVMVolumeGroup resources. The module automatically creates a LVMVolumeGroup resource if it doesn't exist yet for the discovered LVM Volume Group.
- Scanning LVM Physical Volumes on nodes that are part of managed LVM Volume Groups. When underlying block devices are expanded, corresponding LVM Physical Volumes are automatically increased (performs pvresize).
- Creation/expansion/deletion of LVM Volume Groups on the node according to LVMVolumeGroup resource settings.

For more details about the module, refer to the module documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `sds-node-configurator` module and its interactions with other components of DP are shown in the following diagrams:


sds-utils-installer
[Container]
thin-volumes-enabler
[Container]
sds-node-configurator-agent
[Container]
Manages CRs related to block devices and LVM [6443 TCP]
kube-apiserver
[Container]
kube-apiserver [pod]
Kubernetes cluster's control plane
Kubernetes & Scheduling subsystem
Forwards authorized requests [9000 TCP]
Performs request authorization
kube-rbac-proxy (sds-node-configurator)
[Container]
Watches for CRs related to block devices and LVM, updates their statuses [6443 TCP]
Collects metrics [6443 TCP]
sds-node-configurator [pod]
Validates LVMLogicalVolumeSnapshot via webhook svc [8443 TCP]
controller (sds-node-configurator)
[Container]
Collects metrics [4202 TCP]
webhook (sds-node-configurator)
[Container]
prometheus (main)
[Container]
controller (sds-node-configurator) [pod]
prometheus-main [pod]
sds-node-configurator module
prometheus module
Storage subsystem
Observability subsystem
Deckhouse Platform
[Software System]

Module components

The module consists of the following components:

1. **Sds-node-configurator** (DaemonSet): A controller running on cluster nodes and performing mentioned above operations with BlockDevice, LVMVolumeGroup, LVMLogicalVolume, LVMLogicalVolumeSnapshot custom resources, etc. The full list of resources managed by the module is provided in the module documentation.

   It consists of the following containers:

   - **sds-utils-installer**: Init container that installs a set of utilities necessary for managing block devices and LVM volumes.
   - **thin-volumes-enabler**: Init container that enables thin volumes.
   - **sds-node-configurator-agent**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the controller metrics. It is an open-source project.

2. **Controller** (Deployment): A controller that monitors custom resources related to block devices and LVM. The controller works with the metadata of resources and updates their statuses.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: Sidecar container that implements a webhook server for LVMLogicalVolumeSnapshot custom resources validation. If used edition of DP does not support LVM logical volume snapshots functionality, LVMLogicalVolumeSnapshot custom resource does not pass validation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Reconciles custom resources related to block devices and LVM.
   - Authorizes requests for metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Validates LVMLogicalVolumeSnapshot custom resources.
2. **Prometheus-main**: Collects `sds-node-configurator` metrics.

Additional resources

- Module `sds-node-configurator` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-ceph module

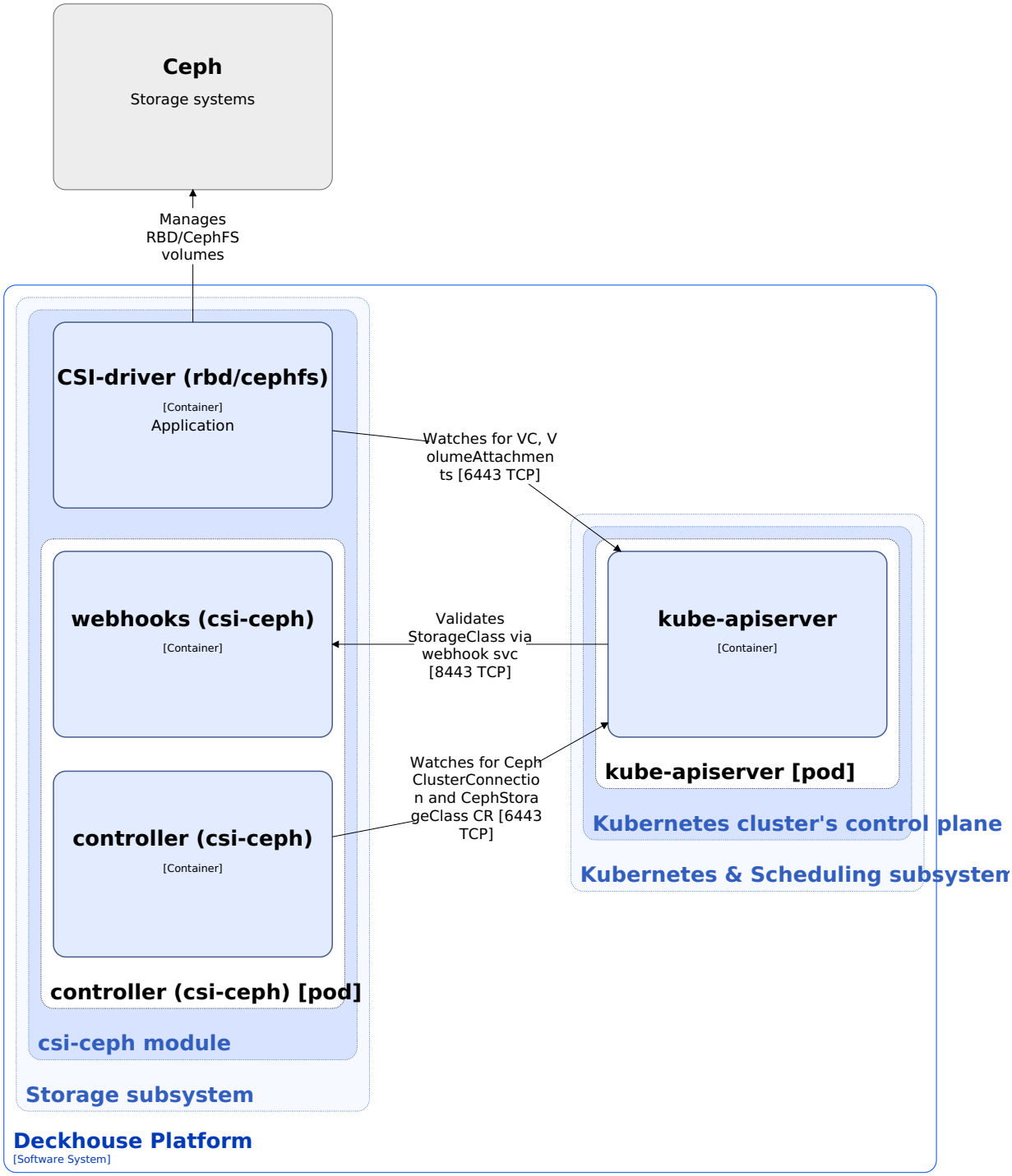
The `csi-ceph` module is designed to integrate DP with Ceph clusters and provides storage management based on RBD (RADOS Block Device) or CephFS. It allows creating StorageClasses in Kubernetes using the CephStorageClass resource.

For more details about the module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-ceph` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:

   - CephClusterConnection: Ceph cluster connection parameters.
   - CephStorageClass: Defines the configuration for the Kubernetes StorageClass that will be created.

   CephStorageClass specifies the storage class type (`CephFS` or `RBD`), reclaim policy, Ceph cluster connection parameters, and additional parameters specific to each storage class type. Depending on the storage class type, these parameters are used by the provisioner of either the `rbd.csi.ceph.com` or `cephfs.csi.ceph.com` CSI driver when managing volumes.

   The CephMetadataBackup custom resource is used in migration and recovery workflows implemented by module hooks, rather than by the main runtime controller.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: A sidecar container that implements a webhook server for StorageClass resources validation.

2. **CSI driver (rbd/cephfs)**: An implementation of the CSI driver for the `rbd.csi.ceph.com` or `cephfs.csi.ceph.com` provisioner. The CSI driver is selected by setting the storage class type in the

CephStorageClass custom resource.

The `csi-cephfs` CSI driver follows the standard CSI driver architecture used in DP.

The `csi-rbd` CSI driver uses an architecture that differs from the standard CSI driver architecture.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
  - Reconciles CephClusterConnection and CephStorageClass custom resources.
  - Creates StorageClass resources.

The following external components interact with the module:

- **Kube-apiserver**: Validates StorageClass resources.

Additional resources

- Module `csi-ceph` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-hpe module

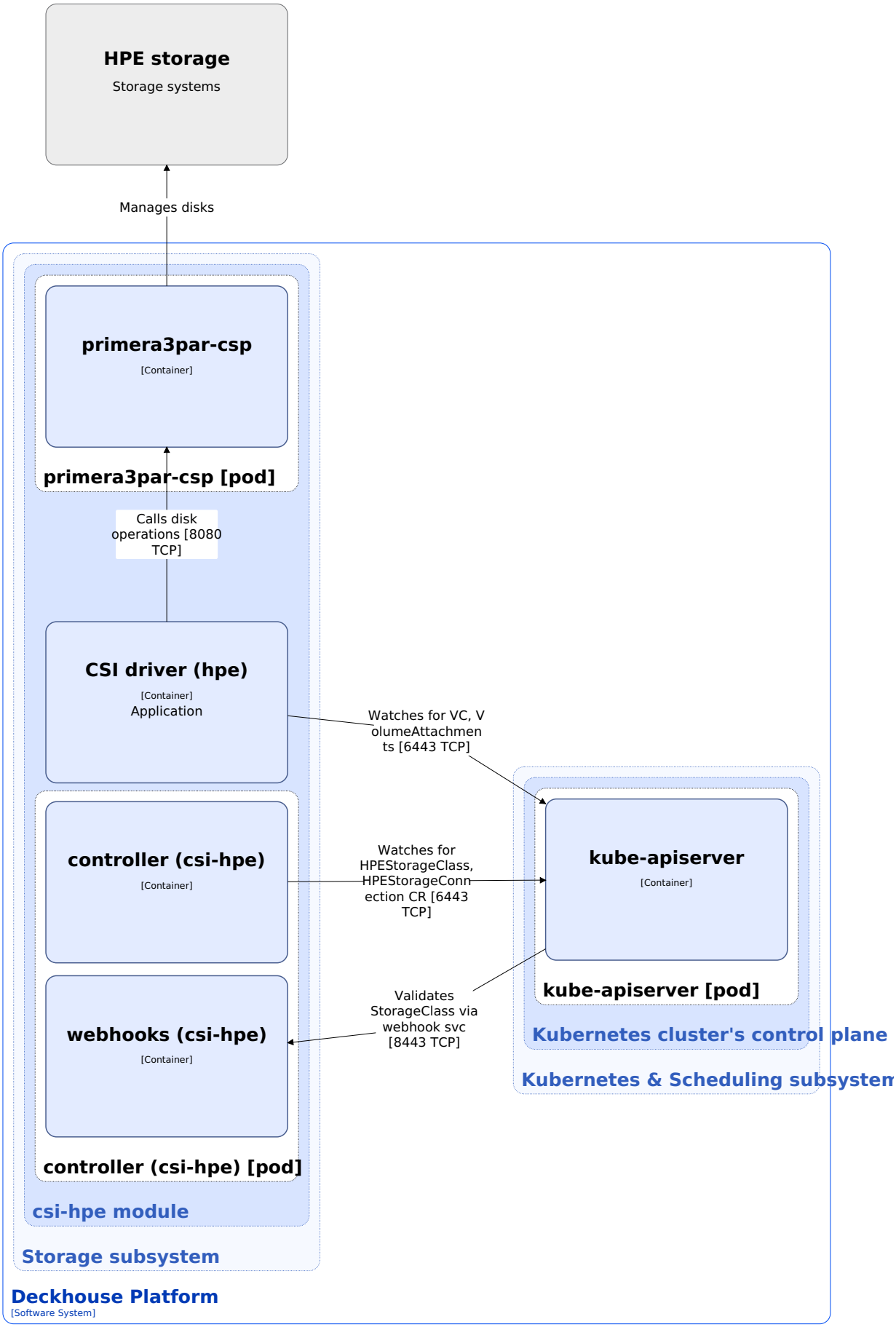
The `csi-hpe` module is designed to manage volumes using HPE storage systems. It enables creating StorageClass resources in Kubernetes using the HPEStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-hpe` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - HPEStorageConnection: Parameters for connecting to HPE storage systems.

    - HPEStorageClass: Defines configuration for Kubernetes StorageClass.

  HPEStorageClass defines the connection protocol, resource pool name, filesystem type, and reclaim policy.

  The controller also synchronizes the `storage.deckhouse.io/csi-hpe-node` label on cluster nodes according to the `spec.settings.nodeSelector` value in the ModuleConfig custom resource.

  It consists of the following containers:

    - **controller**: Main container.
    - **webhooks**: Sidecar container implementing a webhook server for StorageClass validation.

2. **CSI driver (hpe)**: CSI driver implementation for the `csi.hpe.com` provisioner. To study the typical CSI driver architecture used in DP, refer to the CSI driver documentation page.
3. **Primera3par-csp**: A service container provider (Container Storage Provider, CSP) responsible for integrating the CSI driver with HPE Primera and 3PAR storage systems. It handles communication between Kubernetes and the storage arrays, manages session handling, and path replication.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
    - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
    - Reconciles HPEStorageConnection and HPEStorageClass custom resources.
    - Creates and updates VolumeSnapshotClass, Secret, and StorageClass resources.
2. **HPE storage system**: Creates, deletes, and manages volumes, and provides multipath data access.

The following external components interact with the module:

1. **Kube-apiserver**: Validates StorageClass resources.

Additional resources

- Module `csi-hpe` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-huawei module

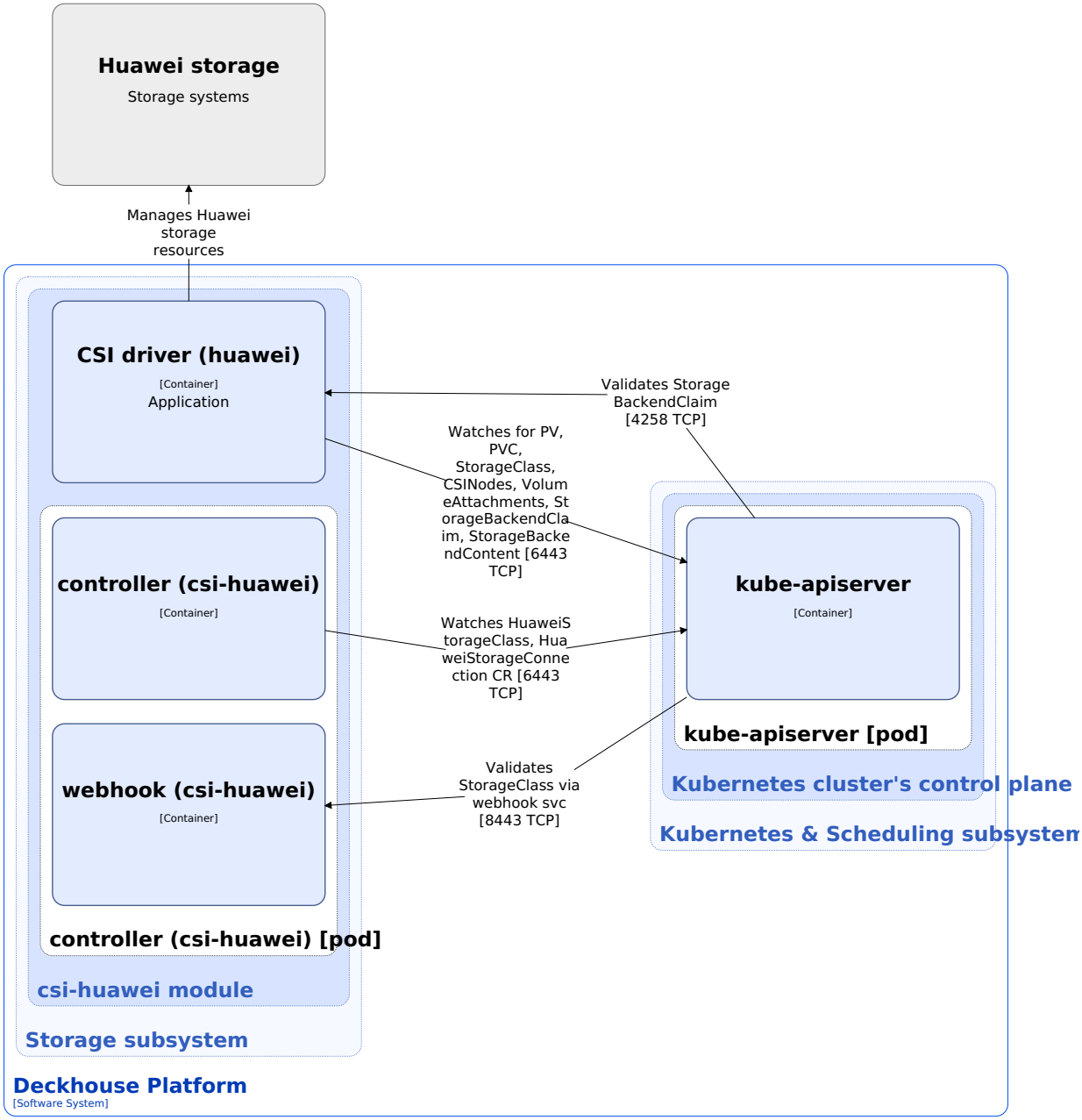
The `csi-huawei` module is designed to manage volumes using Huawei storage systems. It enables creating StorageClass resources in Kubernetes using the HuaweiStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-huawei` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - HuaweiStorageConnection: Parameters for connecting to Huawei storage systems.
   - HuaweiStorageClass: Defines configuration for Kubernetes StorageClass.

   HuaweiStorageClass defines the resource pool name, filesystem type, and reclaim policy.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: Sidecar container implementing a webhook server for StorageClass validation.
2. **CSI driver (huawei)**: CSI driver implementation for the `csi.huawei.com` provisioner. To study the architecture of the `csi-huawei` CSI driver, refer to the corresponding documentation page.

   The `csi-huawei` CSI driver reconciles the following custom resources:

   - StorageBackendClaim: A request to connect to Huawei storage.
   - StorageBackendContent: Description of the actual connection to Huawei storage.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
    - Reconciles HuaweiStorageConnection, HuaweiStorageClass, StorageBackendClaim, and StorageBackendContent custom resources.
    - Creates StorageClass resources.

2. **Huawei storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates StorageBackendClaim custom resources and standard StorageClass resources.

Additional resources

- Module `csi-huawei` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-netapp module

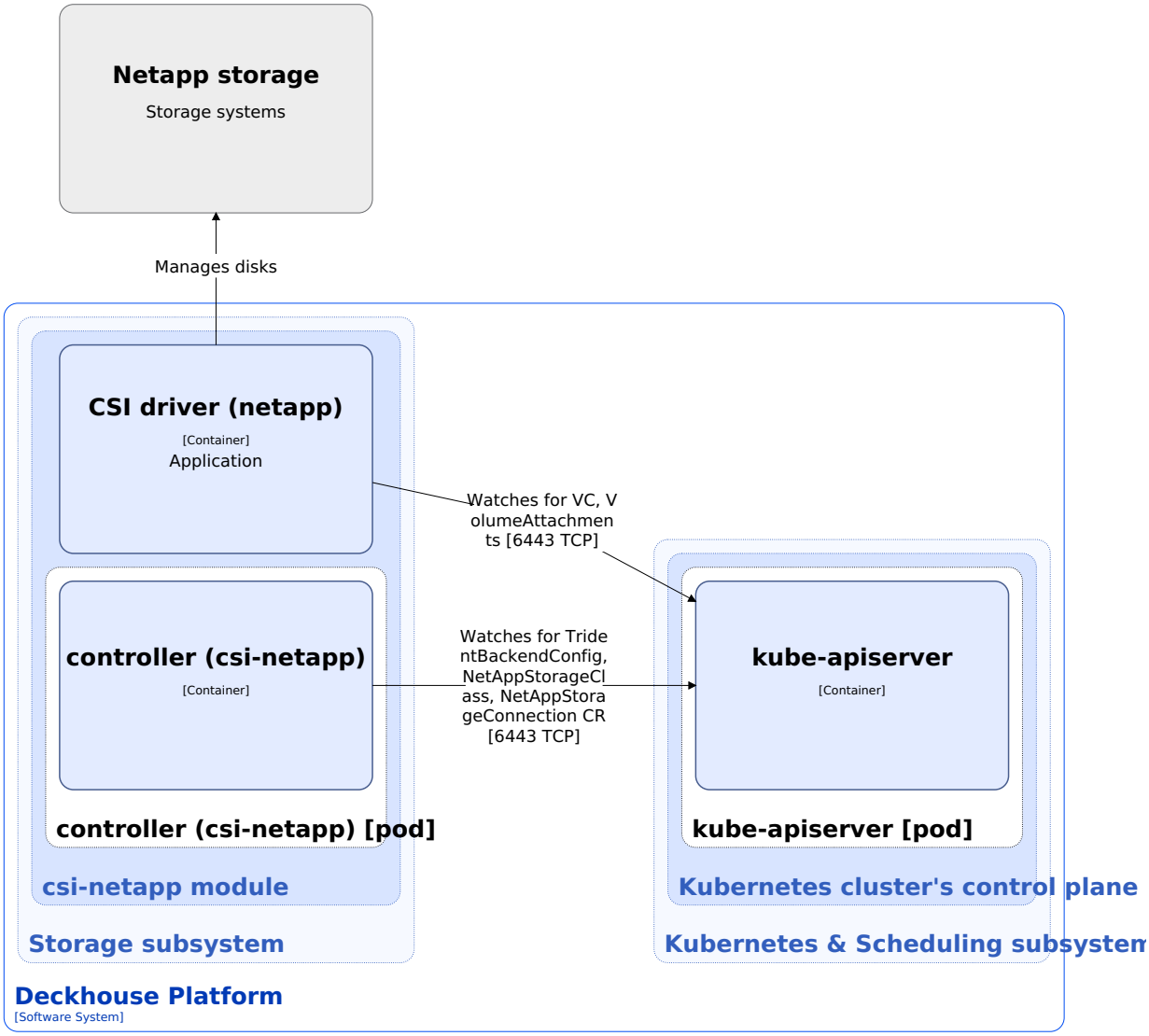
The `csi-netapp` module is designed to manage volumes using NetApp storage systems. It enables creating StorageClass resources in Kubernetes using the NetappStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-netapp` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - NetappStorageConnection: Parameters for connecting to NetApp storage systems.
   - NetappStorageClass: Defines configuration for creating Kubernetes StorageClass that uses the `csi.trident.netapp.io` provisioner.

   The controller also synchronizes the `storage.deckhouse.io/csi-netapp-node` label on cluster nodes according to the `spec.settings.nodeSelector` value in the ModuleConfig custom resource.

   It consists of a single container, **controller**.
2. **CSI driver (netapp)**: CSI driver implementation for the `csi.trident.netapp.io` provisioner. To study the typical CSI driver architecture used in DP, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles TridentBackendConfig, NetappStorageConnection, and NetappStorageClass custom resources.
   - Creates VolumeSnapshotClass, Secret, and StorageClass resources.
2. **NetApp storage system**: Creates, deletes, and manages volumes, and attaches/detaches volumes to/from nodes.

Additional resources

- Module `csi-netapp` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-nfs module

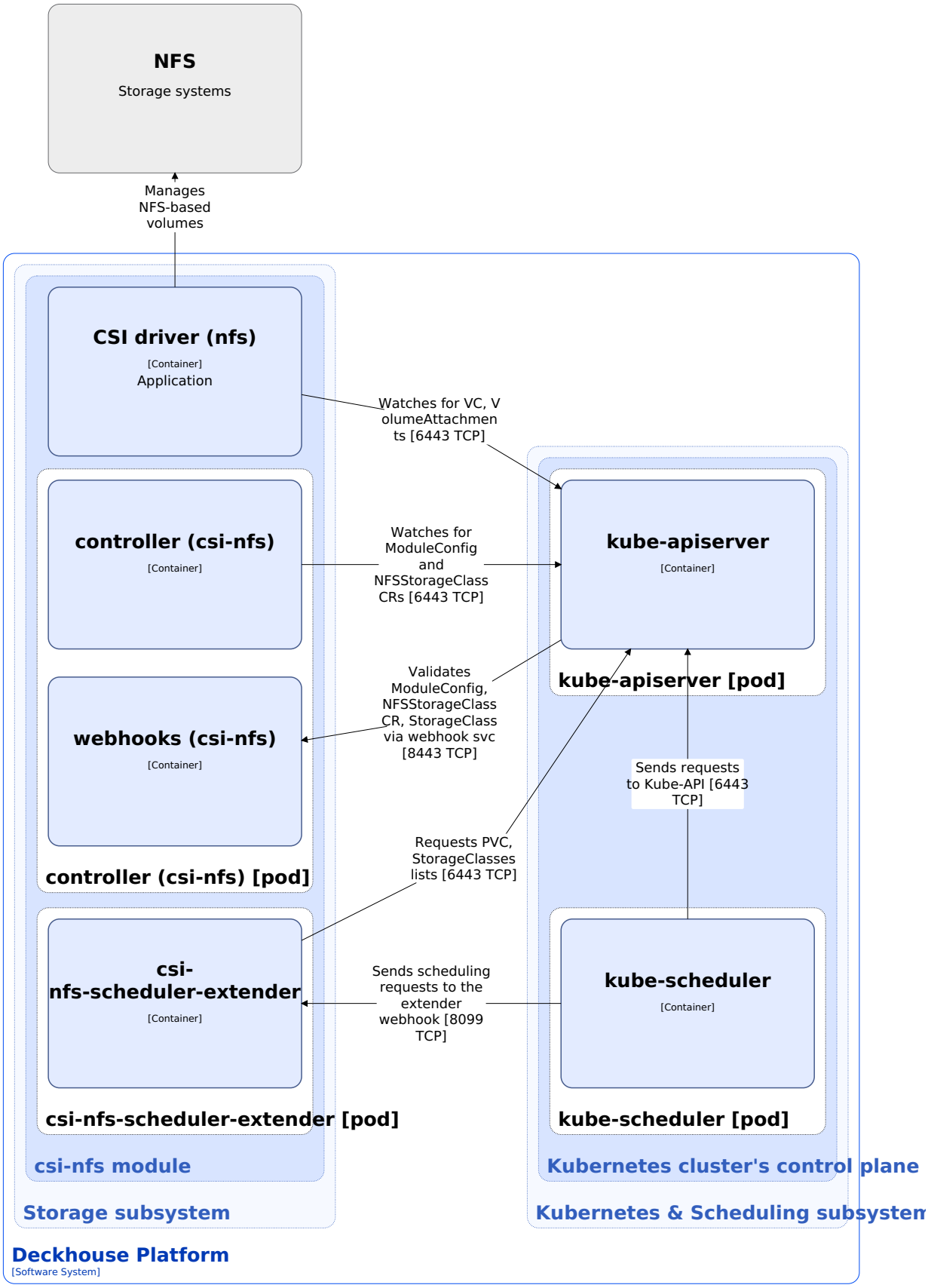
The `csi-nfs` module is designed to manage NFS-based volumes. It enables creating StorageClasses in Kubernetes using the NFSStorageClass resource.

For more details about the module, refer to the module documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-nfs` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Controller**: It reconciles NFSStorageClass custom resources. NFSStorageClass defines the configuration for the created Kubernetes StorageClass, which uses the `nfs.csi.k8s.io` provisioner.

   The created StorageClass configures the NFS server connection settings, a reclaim policy, the volume binding mode, etc. These settings are used by the provisioner of the `csi-nfs` CSI driver when managing NFS-based volumes.

   Also the controller sets the `storage.deckhouse.io/csi-nfs-node` label on the cluster nodes according to the

`spec.workloadNodes.nodeSelector` parameter value of the NFSStorageClass custom resource.

It consists of the following containers:

- **controller**: Main container.
- **webhooks**: A sidecar container that implements a webhook server for validating ModuleConfig and NFSStorageClass custom resources, as well as StorageClass resources.

2. **Csi-nfs-scheduler-extender**: It consists of a single container. It is a kube-scheduler extender, which implements a scheduling logic specific for pods using NFS-based volumes. When scheduling pods, it relies on the node selectors set in the NFSStorageClass custom resource.

   The **csi-nfs-scheduler-extender** may be absent if node selectors are not set in the NFSStorageClass custom resource.

3. **CSI driver (`csi-nfs`)**: It is an implementation of the CSI driver for `nfs.csi.k8s.io` (NFS CSI driver). To study the `csi-nfs` CSI driver architecture, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles NFSStorageClass custom resources.
   - Creates StorageClass resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates ModuleConfig and NFSStorageClass custom resources, as well as StorageClass resources.
2. **Kube-scheduler**: Sends scheduling requests to the `csi-nfs-scheduler-extender` webhook for the pods using NFS-based volumes.

Additional resources

- Module `csi-nfs` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-s3 module

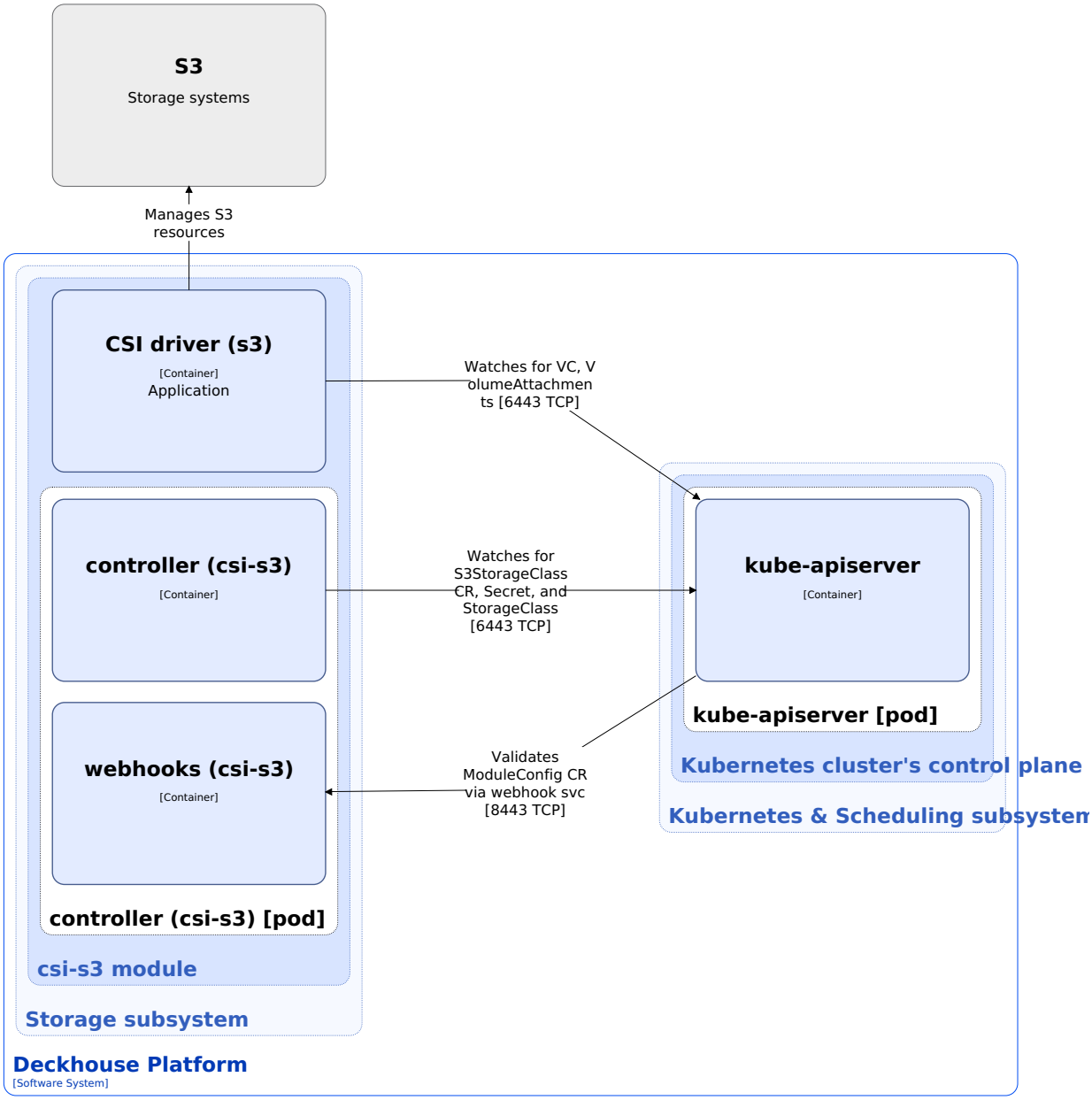
The `csi-s3` module is designed to manage volumes based on S3 storage. The module uses geeseFS, an S3-backed FUSE filesystem. The module enables creating StorageClass resources in Kubernetes using the S3StorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-s3` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the S3StorageClass custom resource. The S3StorageClass resource defines configuration for a Kubernetes StorageClass that uses the `ru.yandex.s3.csi` provisioner.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: Sidecar container implementing a webhook server for ModuleConfig custom resource validation.

2. **CSI driver (s3)**: CSI driver implementation for the `ru.yandex.s3.csi` provisioner. To study the architecture of the `csi-s3` CSI driver, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, Secret, and StorageClass resources.
   - Reconciles S3StorageClass and ModuleConfig custom resources.
   - Creates StorageClass and Secret resources.

2. **S3 storage**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates ModuleConfig custom resources.

Additional resources

- Module `csi-s3` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-scsi-generic module

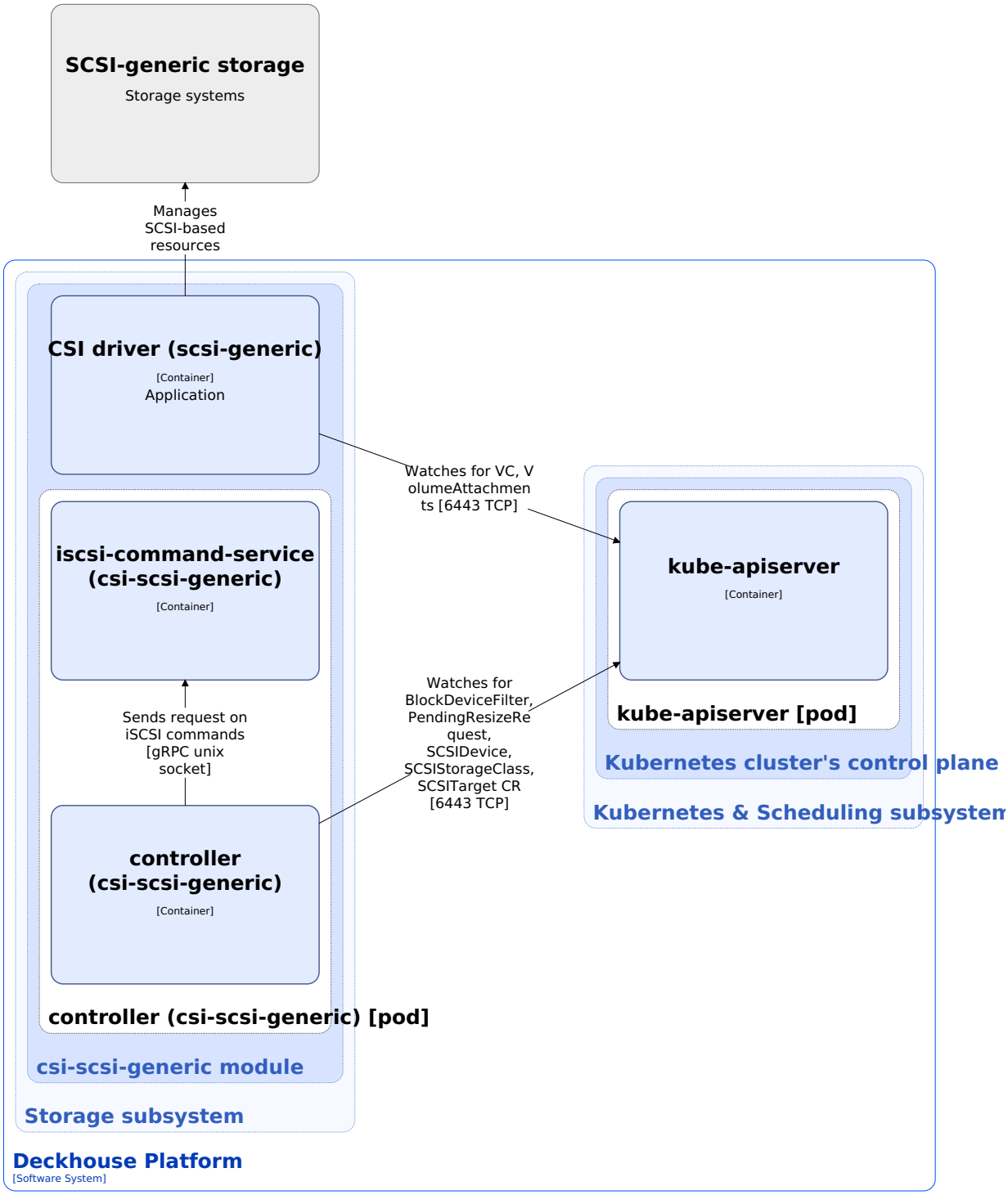
The `csi-scsi-generic` module is designed to manage volumes on storage systems connected over SCSI. It enables creating StorageClass resources in Kubernetes using the SCSIStorageClass resource and supports both iSCSI and Fibre Channel (FC) technologies.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-scsi-generic` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - SCSITarget: Describes a storage connection endpoint (iSCSI/FC).
   - SCSIDevice: Describes a discovered SCSI device.
   - PendingResizeRequest: Represents a request for deferred PVC expansion when the requested size is larger than the current device size.
   - SCSIStorageClass: Defines configuration for Kubernetes StorageClass.
   - BlockDeviceFilter: Specifies filters for selecting physical SCSI devices that can be used to create volumes in the system. Provides flexible management of the available device pool by setting rules based on device attributes and including/excluding specific devices.

   SCSIStorageClass defines the device selector (`scsiDeviceSelector`), reclaim policy, and volume cleanup parameters.

   It consists of the following containers:

   - **controller**: Main container.
   - **iscsi-command-service**: Sidecar container implementing SCSI device discovery.

2. **CSI driver (`csi-scsi-generic`)**: CSI driver implementation for the `scsi-generic.csi.storage.deckhouse.io` provisioner. To study the architecture of the `csi-scsi-generic` CSI driver, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles BlockDeviceFilter, SCSITarget, SCSIDevice, PendingResizeRequest, and SCSIStorageClass custom resources.
   - Creates StorageClass resources.
2. **SCSI-connected storage systems**: Coordinates the usage of available SCSI devices, including their binding and cleanup, as well as node attachment.

Additional resources

- Module `csi-scsi-generic` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-vsphere module

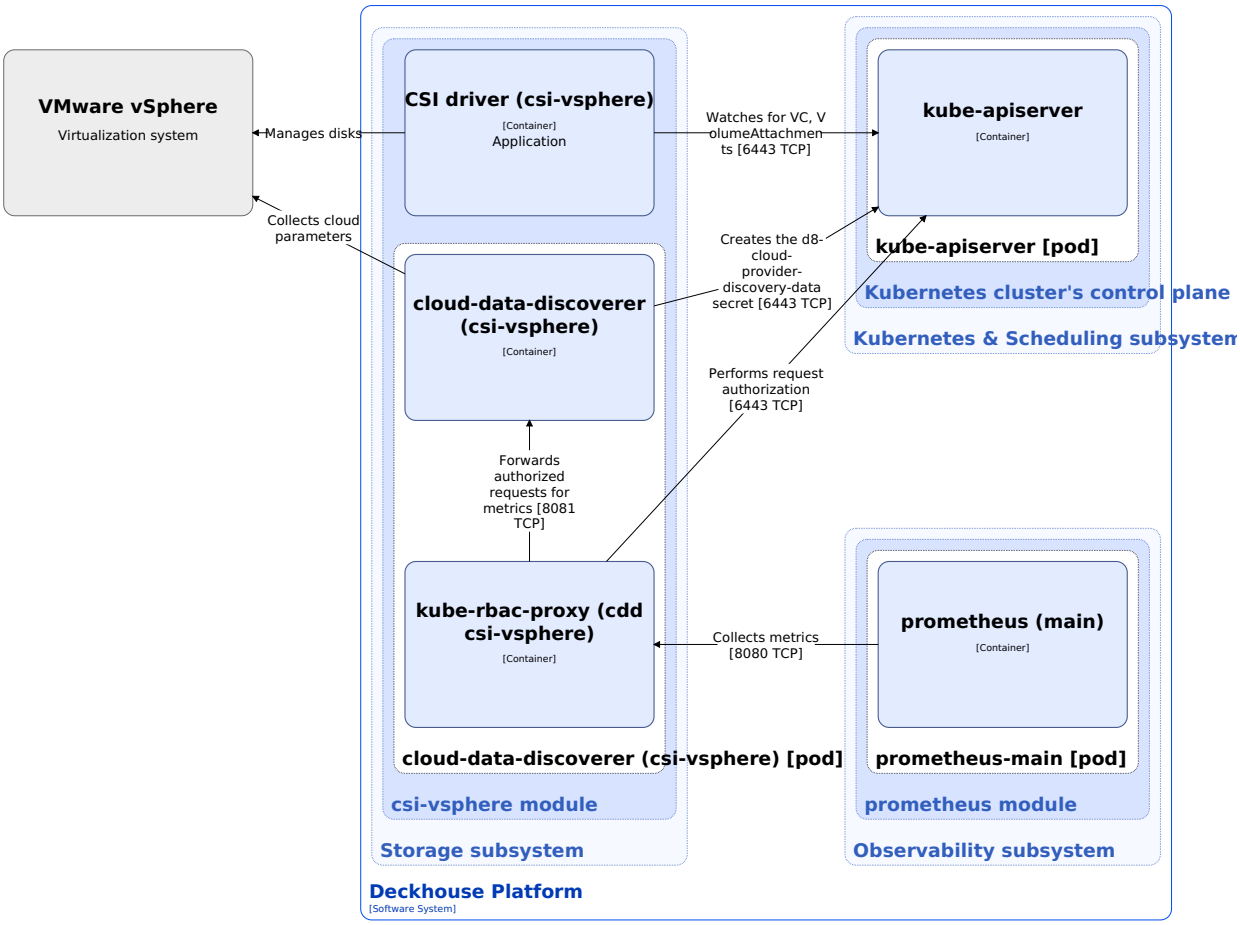
The `csi-vsphere` module provides Container Storage Interface (CSI) support for VMware vSphere environments, enabling dynamic provisioning and management of persistent storage volumes in Kubernetes clusters running on vSphere infrastructure.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-vsphere` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-data-discoverer**: It is responsible for collecting data from the VMware vSphere API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used CSI driver for volume management.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

2. **CSI driver (vsphere)**: It is an implementation of the CSI driver for VMware vSphere. To study the `csi-vsphere` CSI driver architecture, refer to the corresponding documentation section.

   CSI driver (vsphere) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Creates StorageClass resources for each discovered datastore.
   - Authorizes the requests for metrics.

2. **VMware vSphere**:

   - Collects cloud parameters.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Additional resources

- Module `csi-vsphere` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Csi-yadro-tatlin-unified module

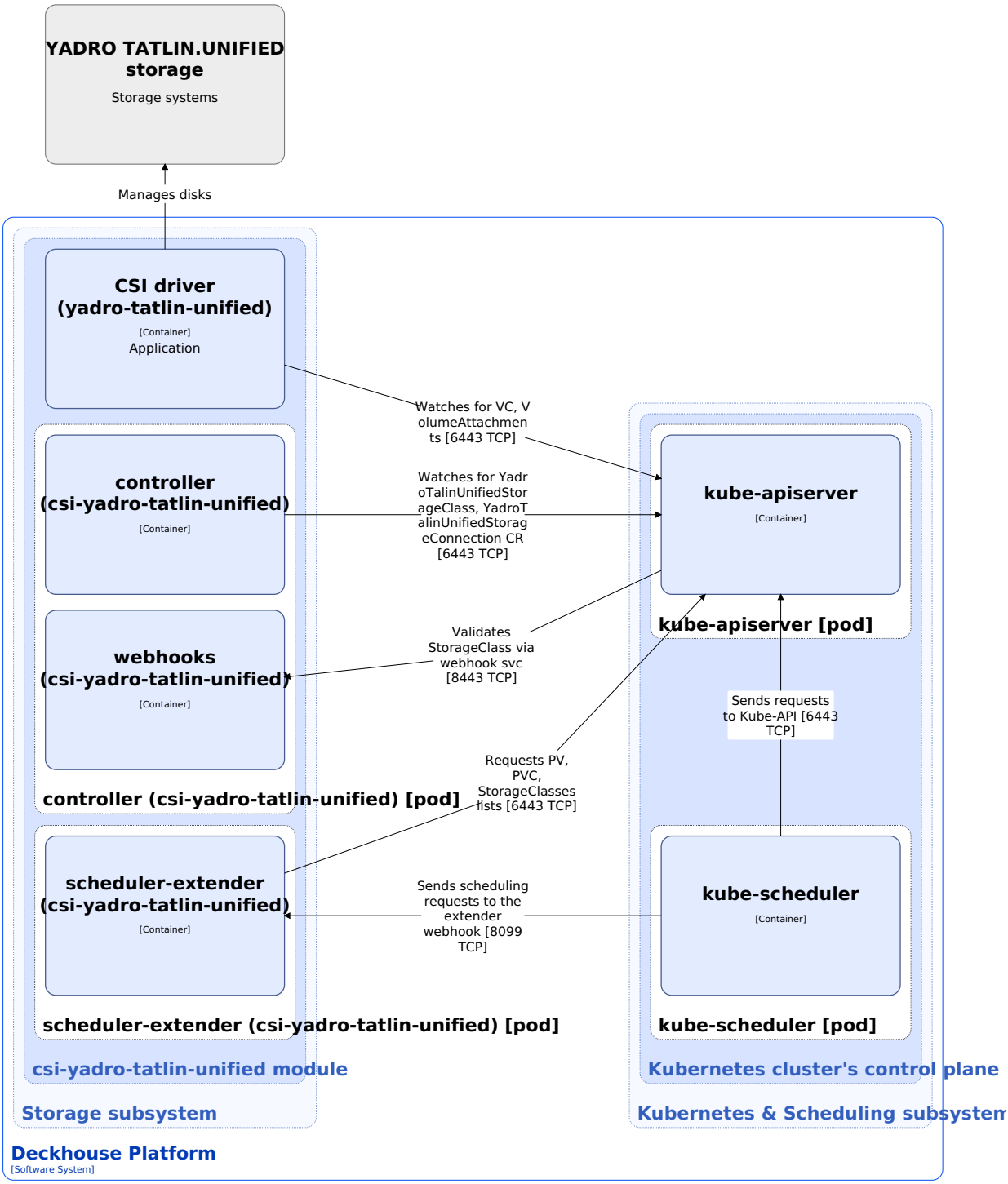
The `csi-yadro-tatlin-unified` module is designed to manage volumes using TATLIN.UNIFIED storage systems. It enables creating StorageClass resources in Kubernetes using the YadroTatlinUnifiedStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-yadro-tatlin-unified` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:

   - YadroTatlinUnifiedStorageConnection: Parameters for connecting to TATLIN.UNIFIED storage systems.
   - YadroTatlinUnifiedStorageClass: Defines configuration for creating Kubernetes StorageClass that uses the `csi-tatlinunified.yadro.com` provisioner.

   YadroTatlinUnifiedStorageClass defines connection parameters, as well as resource pool name, filesystem type, and reclaim policy.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: Sidecar container implementing a webhook server for StorageClass resource validation.

2. **CSI driver (yadro-tatlin-unified)**: CSI driver implementation for the `csi-tatlinunified.yadro.com` provisioner. To study the typical CSI driver architecture used in DP, refer to the CSI driver documentation page.

3. **Scheduler-extender**: A single-container component that acts as an extender for kube-scheduler. It implements scheduling logic specific to pods using volumes from TATLIN.UNIFIED storage systems. During scheduling, node

selectors defined in the YadroTatlinUnifiedStorageConnection custom resource under the `controlPlane` and `dataPlane` parameters are taken into account.

The component may be absent if node selectors are not set in the YadroTatlinUnifiedStorageConnection custom resource.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles YadroTatlinUnifiedStorageConnection and YadroTatlinUnifiedStorageClass custom resources.
   - Creates StorageClass resources.
2. **YADRO TATLIN.UNIFIED storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates standard StorageClass resources.

Additional resources

- Module `csi-yadro-tatlin-unified` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Standard CSI driver

A CSI driver (plugin) is used to manage persistent storage volumes in Deckhouse Platform (DP).

Container Storage Interface (CSI) is a standard interface that unifies access to storage systems and simplifies integration of different storage systems into clusters.

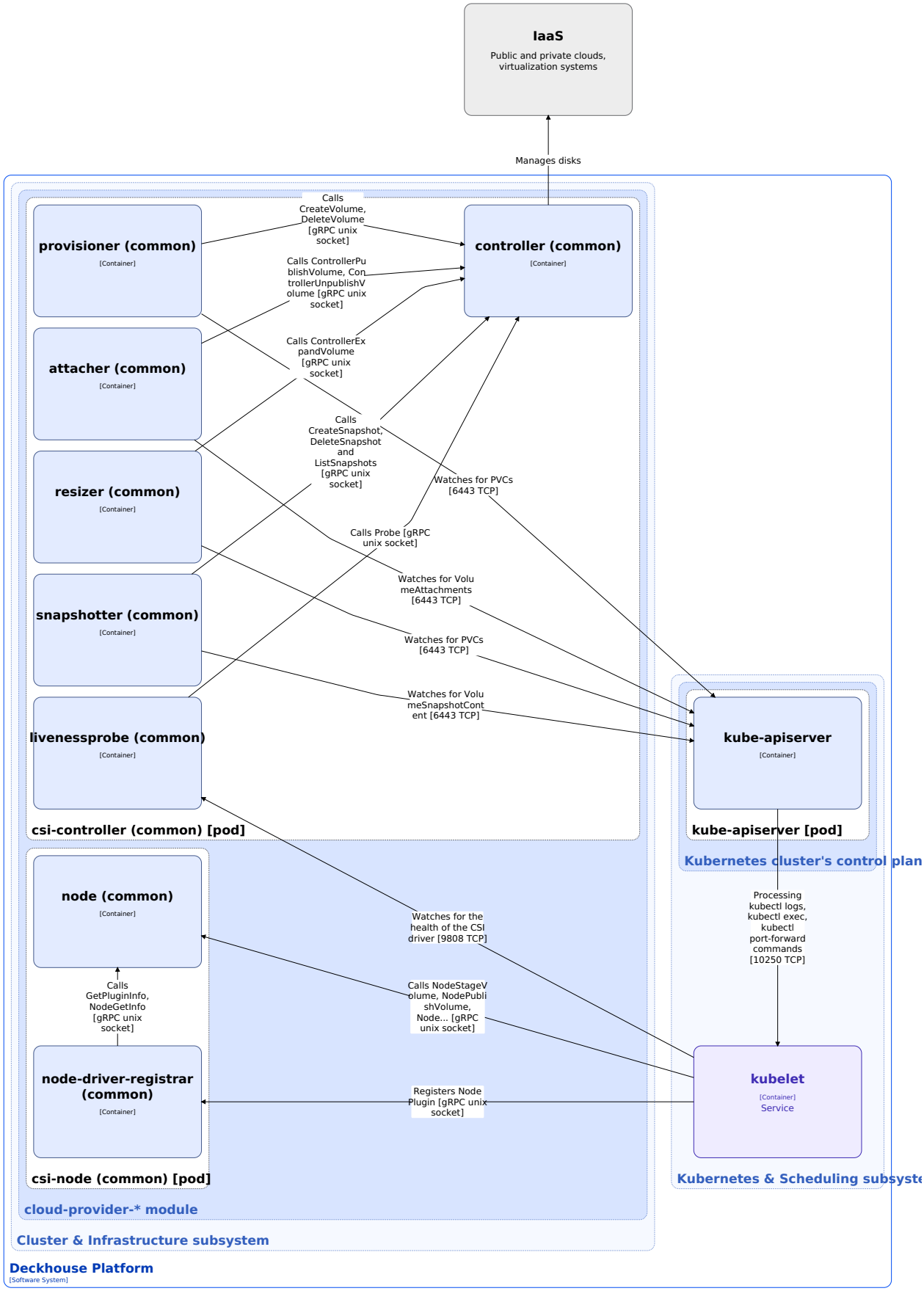
The CSI driver is included in DP modules named `cloud-provider-*` and `csi-*`. Although each supported cloud provider or storage system uses its own implementation of the CSI specification, the architecture of the CSI driver is the same across all implementations. Implementations may differ in the set of supported capabilities and components.

The following is a description of the reference CSI driver architecture used in DP. It includes all possible components and implements the full functionality used by DP modules.

Driver architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The reference CSI driver architecture at Level 2 of the C4 model and its interactions with other DP components are shown in the following diagram:



Driver components

The CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots. For example, in AWS this component calls the EC2 API to create EBS volumes.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services

Identity Service and Controller Service according to the CSI specification.

- **controller sidecar containers**: Kubernetes community-maintained external controllers.

  These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

  External controllers communicate with the controller container over gRPC via Unix sockets.

  Csi-controller includes the following external controllers:

  - **provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
  - **attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
  - **resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
  - **snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
  - **livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

i Some sidecar containers from the external controller list (for example, snapshotter) may be absent in the Deployment of specific `cloud-provider-*` or `csi-*` modules if the corresponding functionality is not supported by the CSI driver implementation.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **Cloud infrastructure** (or a virtualization system): Creates and deletes volumes, attaches and detaches volumes from nodes, and manages snapshots.

The following external components interact with the driver:

1. Kubelet:
   - Checks the CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

   Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

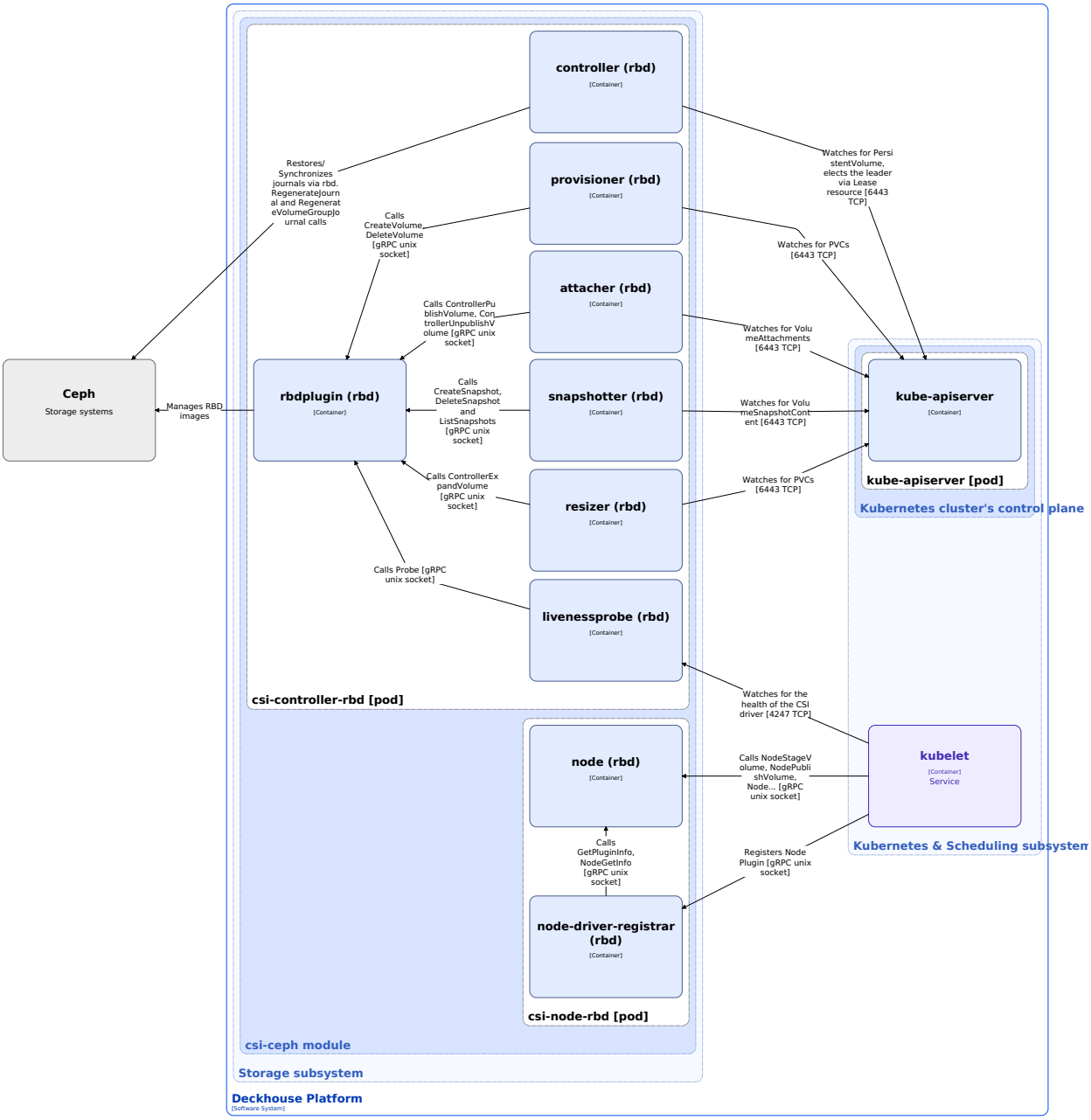
Ceph RBD CSI driver

The `csi-rbd` CSI driver is an implementation of the Container Storage Interface (CSI) standard to manage Ceph RBD (RADOS Block Device) volumes in Deckhouse Platform (DP).

Driver architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-rbd` CSI driver and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Driver components

The `csi-rbd` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container that performs background tasks for managing Kubernetes resources related to Ceph.
   - **rbdplugin**: Sidecar container exposes the CSI endpoint over a Unix socket, implements Identity Service and Controller Service according to the CSI specification. It interacts with the Ceph cluster for RBD volumes.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI RPC methods exposed by the rbdplugin container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the rbdplugin container over gRPC via Unix sockets.

     Csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
     - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the **node** container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **Ceph cluster**: Creates and deletes volumes, attaches and detaches volumes from nodes, and manages snapshots.

The following external components interact with the driver:

- Kubelet:
  - Checks the CSI driver livenessProbe.
  - Registers the Node Plugin.
  - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

  Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

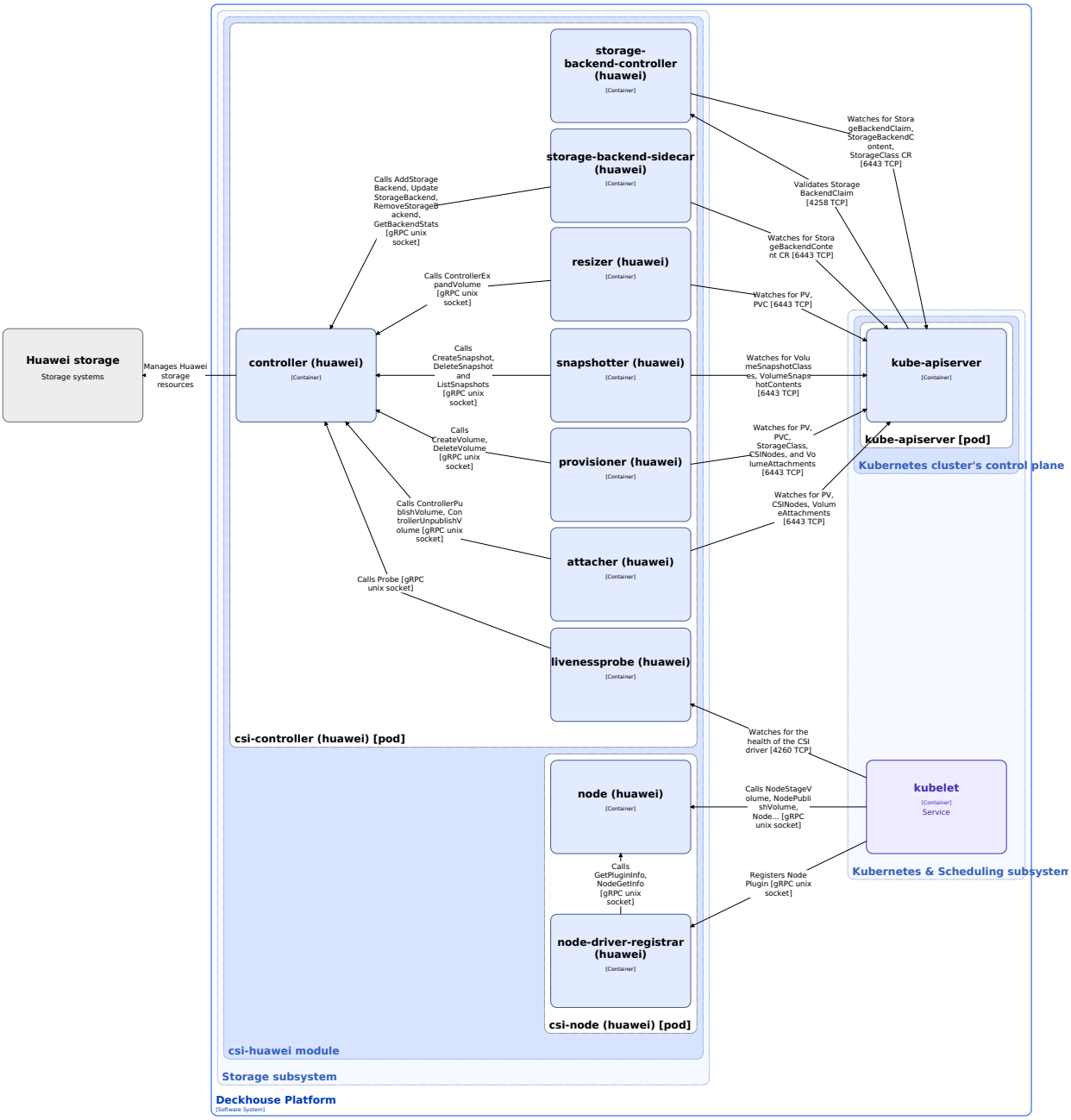
Huawei CSI driver

The `csi-huawei` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage volumes on Huawei storage systems in Deckhouse Platform (DP).

Driver architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-huawei` CSI driver and its interactions with other components of DP are shown in the following diagram:



Driver components

The `csi-huawei` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
     - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots`.
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.
   - **storage-backend-controller**: An additional controller that reconciles the following custom resources:
     - StorageBackendClaim: A request to connect to Huawei storage.
     - StorageBackendContent: Description of the actual connection to Huawei storage.
   - **storage-backend-sidecar**: Sidecar container that provides integration with Huawei storage via `AddStorageBackend`, `UpdateStorageBackend`, `RemoveStorageBackend`, and `GetBackendStats` RPC calls.
2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolumeClaim and VolumeAttachment resources.
   - Reconciles StorageBackendClaim and StorageBackendContent custom resources.
2. **Huawei storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the driver:

1. **Kube-apiserver**: Validates StorageBackendClaim custom resources.
2. Kubelet:
   - Checks CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the

Node Plugin.

Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

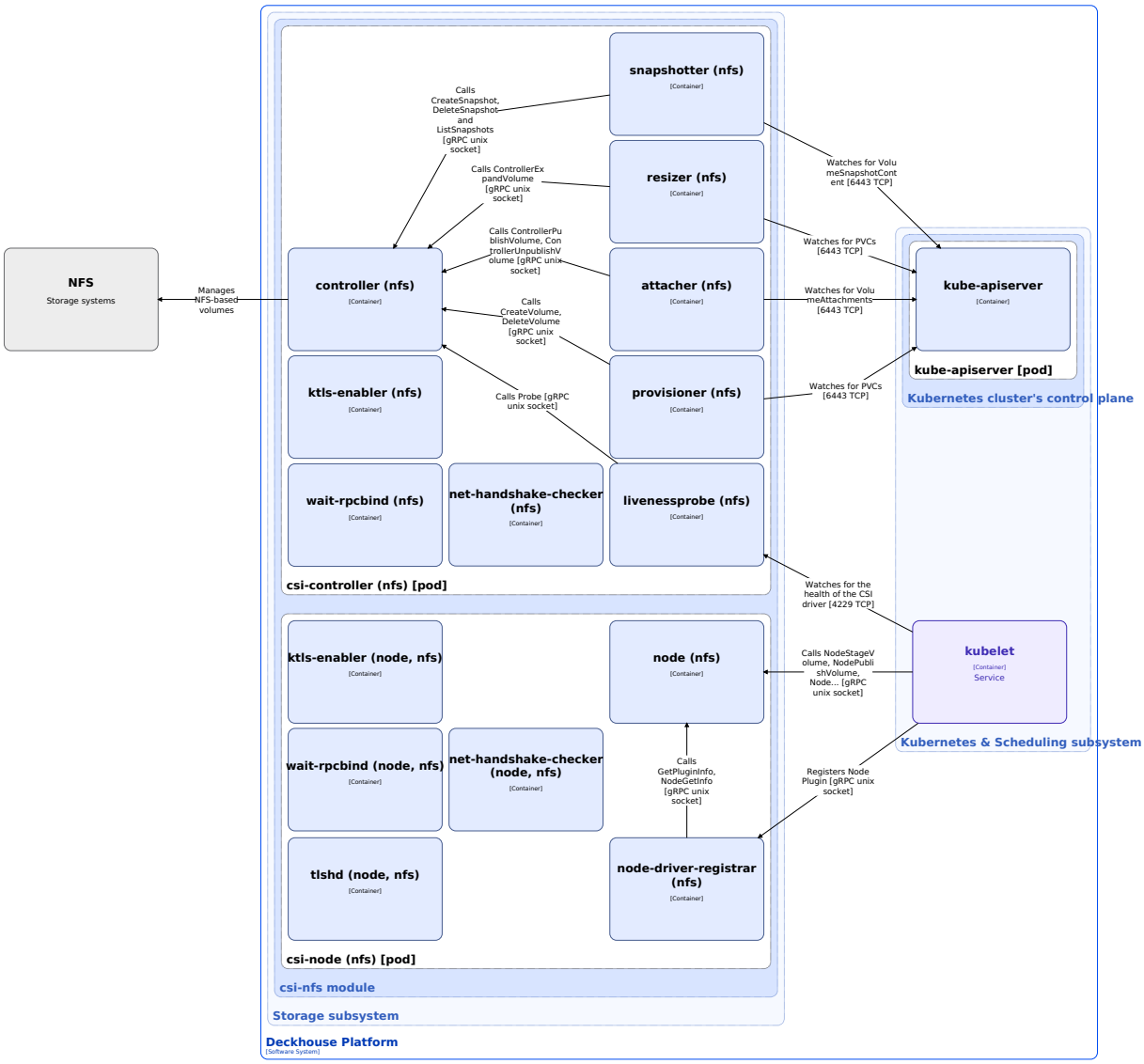
NFS CSI driver

The CSI driver `csi-nfs` is the implementation of the Container Storage Interface (CSI) standard to manage NFS-based volumes in Deckhouse Platform (DP).

Driver architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-nfs` CSI driver and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Driver components

The `csi-nfs` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **wait-rpcbind**: Init-container. It waits for when the Unix socket of the `rpcbind` service is ready and accepts connections.
   - **ktls-enabler**: Init-container. It loads and checks Linux kernel modules on the node for RPC-with-TLS support.
   - **net-handshake-checker**: Init-container. It runs `tlshd` one time to check kernel TLS handshake support.
   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     Csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
     - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on cluster nodes labeled with `storage.deckhouse.io/csi-nfs-node` and responsible for local volume mount and unmount operations.

   > ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **wait-rpcbind**: Init-container. It waits for when the Unix socket of the `rpcbind` service is ready and accepts connections.
   - **ktls-enabler**: Init-container. It loads and checks Linux kernel modules on the node for RPC-with-TLS support.
   - **net-handshake-checker**: Init-container. It runs `tlshd` one time to check kernel TLS handshake support.
   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **tlshd**: Sidecar container that provides an RPC-with-TLS connection.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **NFS storage**:
   - Attaches and detaches NFS-based volumes from nodes.
   - Manages snapshots.

The following external components interact with the driver:

- Kubelet:
  - Checks the CSI driver livenessProbe.
  - Registers the Node Plugin.
  - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

  Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

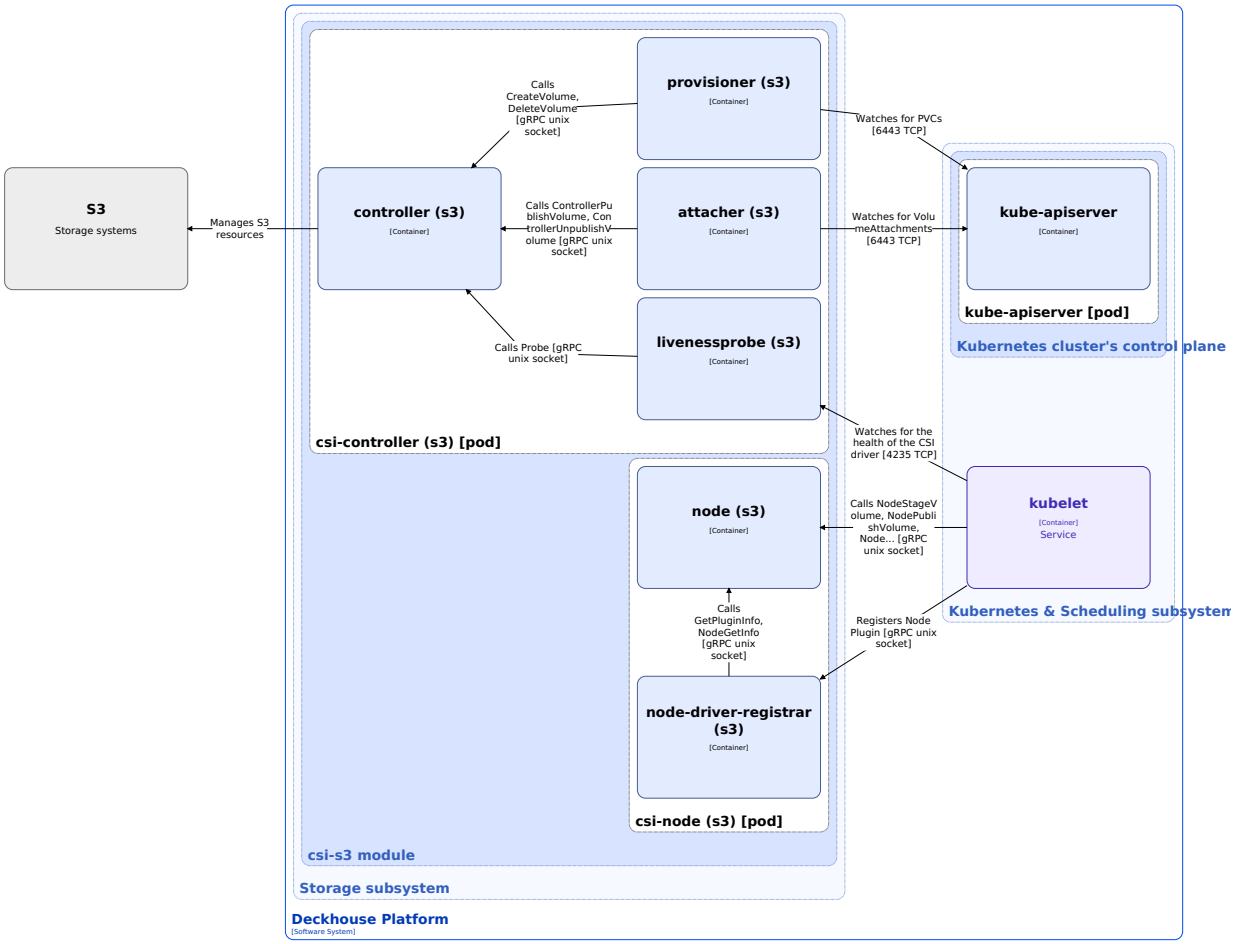
S3 CSI driver

The `csi-s3` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage S3-based volumes in Deckhouse Platform (DP).

Driver architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-s3` CSI driver and its interactions with other components of DP are shown in the following diagram:



Driver components

The `csi-s3` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, and attaching and detaching volumes from nodes.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers interact with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Watches PersistentVolumeClaim and VolumeAttachment resources.
2. **S3 storage**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the driver:

1. Kubelet:

   - Checks CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the Node Plugin.

   Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

> ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

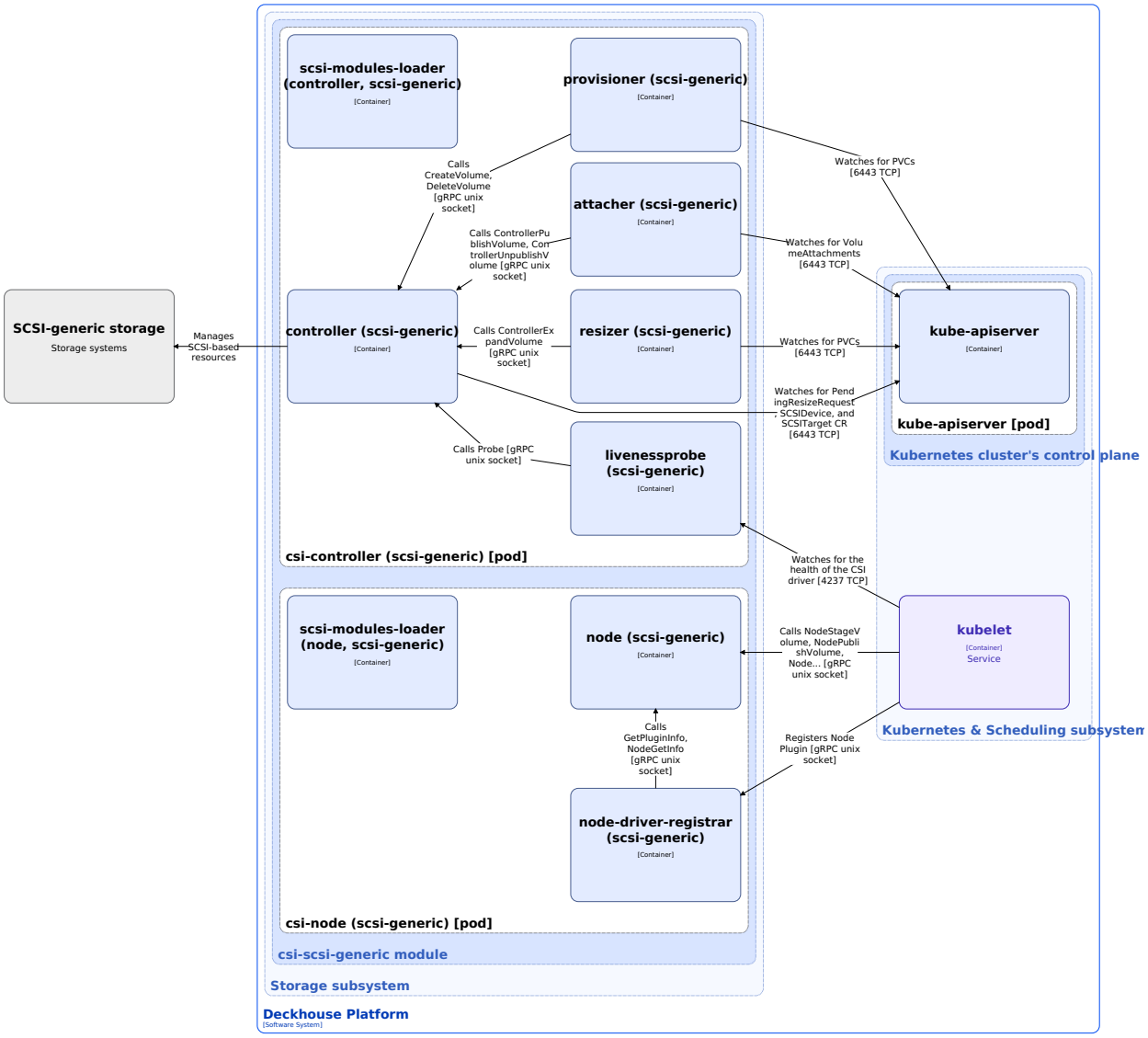
SCSI Generic CSI driver

The `csi-scsi-generic` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage volumes on SCSI-connected storage systems in Deckhouse Platform (DP). The CSI driver supports both iSCSI and Fibre Channel (FC) connection technologies.

Driver architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-scsi-generic` CSI driver and its interactions with other components of DP are shown in the following diagram:



Driver components

The `csi-scsi-generic` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, and attaching and detaching volumes from nodes.

   It consists of the following containers:

   - **scsi-modules-loader**: Init container that loads the Linux kernel module required for iSCSI (`iscsi_tcp`).
   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers interact with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

- **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`.
- **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node.
- **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability. The controller doesn't expand the Logical Unit Number (LUN) size in the storage system, but updates information in the cluster metadata.
- **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **scsi-modules-loader**: Init container that loads the kernel module required for iSCSI (`iscsi_tcp`).
   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, PendingResizeRequest, SCSIDevice, and SCSITarget resources.
2. **SCSI-connected storage systems**: Coordinates the usage of already available SCSI devices, including their binding/cleanup and node attachment.

The following external components interact with the driver:

1. Kubelet:
   - Checks CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the Node Plugin.

   Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

VMware vSphere CSI driver

A CSI driver (plugin) is used to manage persistent storage volumes in Deckhouse Platform (DP).

Container Storage Interface (CSI) is a standard interface that unifies access to storage systems and simplifies integration of different storage systems into clusters.

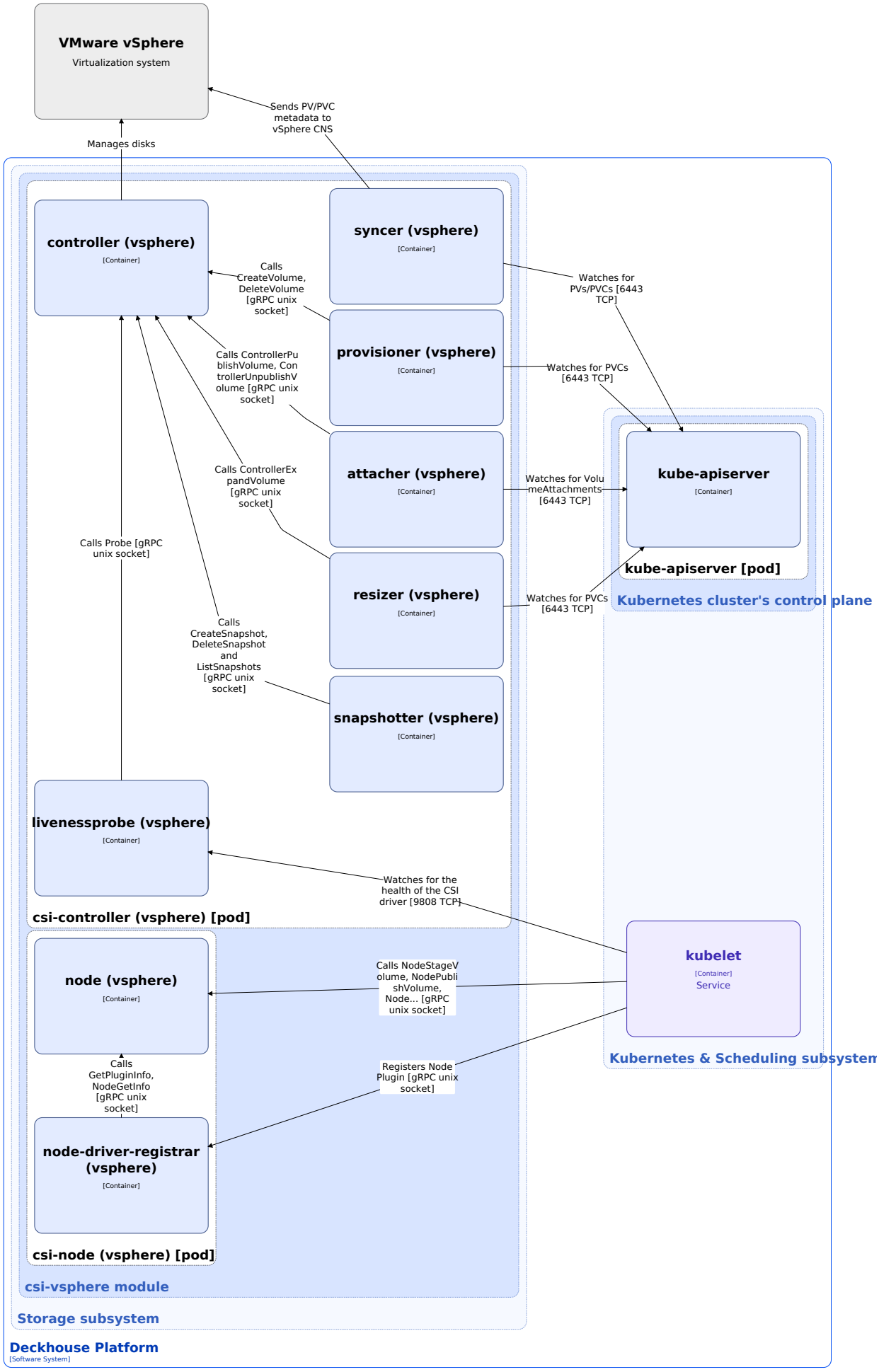
The `cloud-provider-vsphere` module uses `csi-vsphere` (based on vSphere CSI Driver). It differs from the drivers used in other DP modules by the presence of a syncer component specific to VMware vSphere.

Driver architecture

ℹ The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-vsphere` CSI driver and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Driver components

The CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     Csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
     - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.
     - **Syncer**: VMware vSphere specific controller, it is responsible for pushing the PersistentVolumes, PersistentVolumeClaims, and pod metadata to Cloud Native Storage (CNS). CNS is a control plane managing various volume types in the VMware vSphere storage system.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > **Warning.** The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **VMware vSphere**:
   - Creates and deletes volumes.
   - Attaches and detaches volumes from nodes.
   - Manages snapshots.
   - Synchronizes PersistentVolumes, PersistentVolumeClaims, and pod metadata with the Cloud Native Storage (CNS).

The following external components interact with the driver:

1. Kubelet:
   - Checks the CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

   Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `cloud-provider-vsphere` documentation
- Module `csi-vsphere` documentation
- Module `snapshot-controller` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Observability subsystem

This section describes the architecture of the Observability subsystem of Deckhouse Platform (DP).

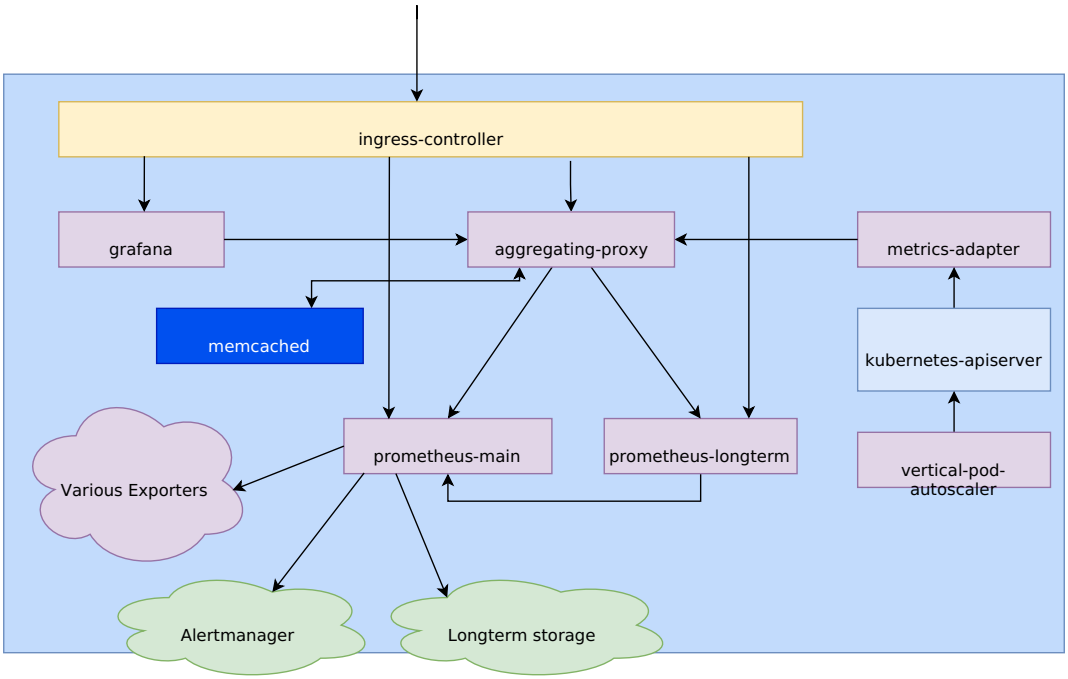
The Observability subsystem includes the following modules:

- `prometheus`: Deploys a monitoring stack with predefined settings for DP and applications, simplifying the initial configuration.
- `operator-prometheus`: Installs Prometheus Operator, which automates the deployment and management of Prometheus instances.
- `prometheus-metrics-adapter`: Allows HPA and VPA autoscalers to use monitoring metrics when making scaling decisions.
- `log-shipper`: Simplifies log collection setup in Kubernetes clusters.
- `loki`: Deploys a short-term log storage system in the cluster based on Grafana Loki.
- `observability`: Extends the functionality of the `prometheus` module and the Deckhouse web UI, providing additional capabilities for flexible management of metrics, dashboards, and alerts, as well as access control mechanisms for them.
- `extended-monitoring`: Enhances cluster monitoring by deploying additional Prometheus exporters that help detect potential issues before they affect service operation.
- `monitoring-custom`: Simplifies monitoring configuration for user applications by requiring only a specific label to be set for the target application.
- `monitoring-kubernetes`: Provides transparent and timely monitoring of all cluster nodes and key infrastructure components.
- `upmeter`: Checks platform availability and cluster component health in real time and displays the results on dedicated dashboards.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Architecture of monitoring in the Deckhouse Platform

Composition and interaction scheme of monitoring components



Components installed by DP

Component	Description
prometheus-operator	DP module responsible for running Prometheus in the cluster.
prometheus-main	Main Prometheus that performs scraping every 30 seconds (can be changed using the `scrapeInterval` parameter). It processes all rules, sends alerts and is the main data source.
prometheus-longterm	Additional Prometheus storing sparse data samples from the main prometheus-main
aggregating-proxy	Aggregating and caching proxy that combines main and longterm into one source. Helps avoid data gaps when one of the Prometheus instances is unavailable.
memcached	In-memory data caching service.
grafana	UI for displaying metrics in dashboard format.
metrics-adapter	Component providing Kubernetes API for accessing metrics. Required for proper VPA operation.
Various exporters	Set of ready-made Prometheus exporters for all necessary metrics: `kube-state-metrics`, `node-exporter`, `oomkill-exporter`, `image-availability-exporter`.
upmeter	Module for assessing DP component availability.
trickster	Caching proxy that reduces load on Prometheus. Will be deprecated soon.

Prometheus

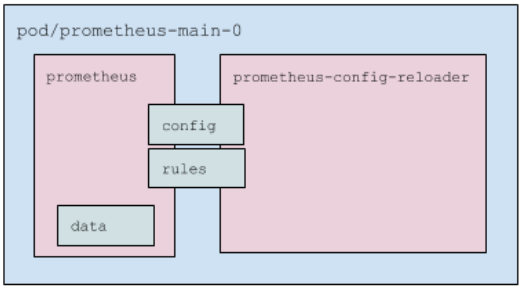
Prometheus collects metrics and executes rules:

- For each *target* (monitoring target) with a given period `scrape_interval`, Prometheus makes an HTTP request to this *target*, receives metrics in its own format in response and saves them to its database.
- Each `evaluation_interval` processes rules (*rules*), based on which:
  - sends alerts;
  - or saves new metrics (result of rule execution) to its database.

Prometheus is installed by the `prometheus-operator` module of DP, which performs the following functions:

- Defines the following custom resources:
  - `Prometheus`: Defines the installation (cluster) of *Prometheus*.
  - `ServiceMonitor`: Defines how to collect metrics from services.
  - `Alertmanager`: Defines the cluster of *Alertmanagers*.
  - `PrometheusRule`: Defines the list of *Prometheus rules*.
- Monitors these resources, and also:
  - Generates `StatefulSet` with *Prometheus* itself.
  - Creates secrets with configuration files necessary for Prometheus operation (`prometheus.yaml` — Prometheus configuration, and `configmaps.json` — configuration for `prometheus-config-reloader`).
  - Monitors `ServiceMonitor` and `PrometheusRule` resources and updates *Prometheus* configuration files by making changes to secrets based on them.

What's in the Prometheus pod?



- Two containers:
  - `prometheus`: Prometheus itself;
  - `prometheus-config-reloader`: Wrapper that:
    - Monitors changes to `prometheus.yaml` and, if necessary, calls configuration reload to Prometheus (via special HTTP request, see more details below).
    - Monitors PrometheusRules (see more details below) and downloads them and restarts Prometheus as needed.
- Pod uses three volumes:
  - config: Mounted secret (two files: `prometheus.yaml` and `configmaps.json`). Connected to both containers.
  - rules: `emptyDir` that is filled by `prometheus-config-reloader` and read by `prometheus`. Connected to both containers, but in `prometheus` in read-only mode.
  - data: Prometheus data. Mounted only in `prometheus`.

How is Prometheus configured?

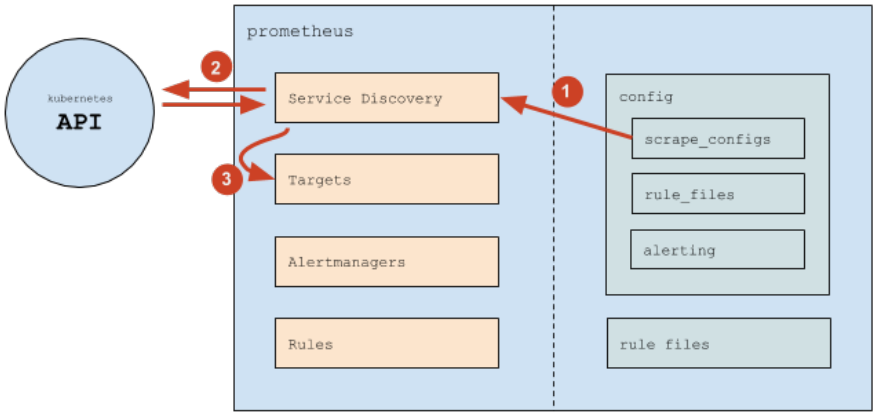
- The Prometheus server has *config* and *rule files* (files with rules);
- The `config` has the following sections:
  - `scrape_configs`: Settings for finding *targets* (monitoring targets, see more details in the next section).
  - `rule_files`: List of directories where *rules* that need to be loaded are located:

    ```
    rule_files:
    - /etc/prometheus/rules/rules-0/*
    - /etc/prometheus/rules/rules-1/*
    ```

  - `alerting`: Settings for finding *Alert Managers* to send alerts to. This section is very similar to `scrape_configs`, only the result of its work is a list of *endpoints* to which Prometheus will send alerts.

Where does Prometheus get the list of *targets*?

- In general, Prometheus works as follows:

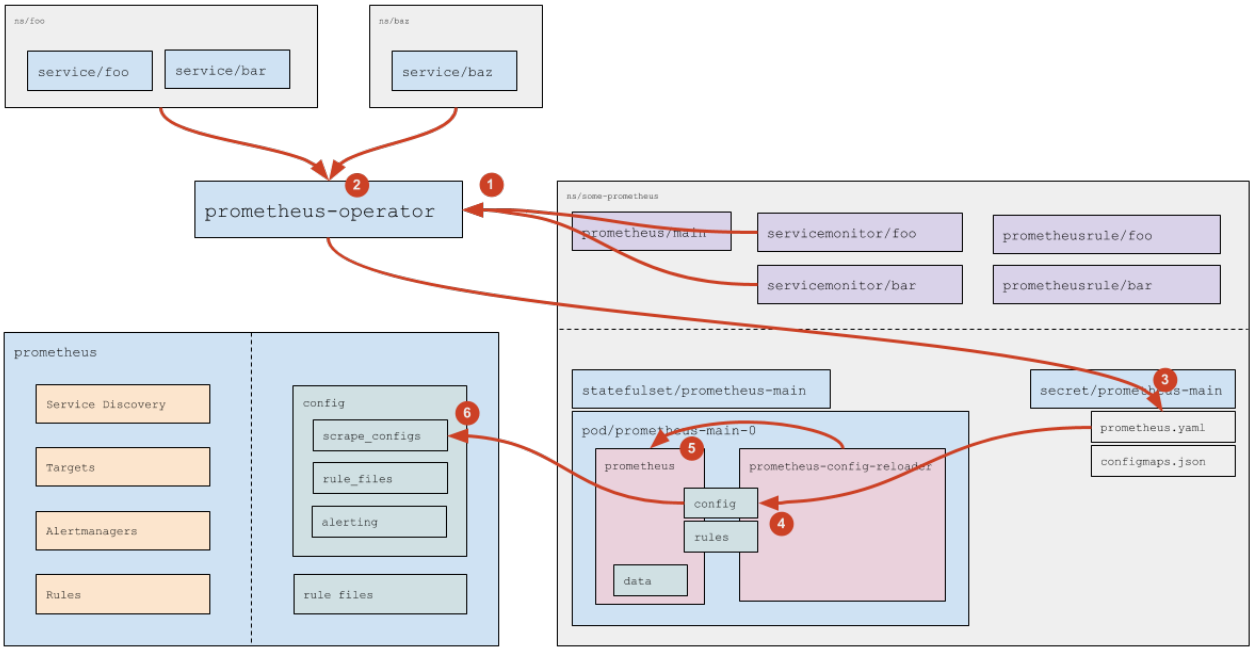


  - **(1)** Prometheus reads the `scrape_configs` section of the config, according to which it configures its internal Service Discovery mechanism.
  - **(2)** The Service Discovery mechanism interacts with the Kubernetes API (mainly — gets endpoints).
  - **(3)** Based on what happens in Kubernetes, the Service Discovery mechanism updates Targets (list of *targets*).
- In `scrape_configs` a list of *scrape jobs* (internal Prometheus concept) is specified, each of which is defined as follows:

```
scrape_configs:
  # General settings
- job_name: d8-monitoring/custom/0   # just the name of the scrape job, shown in the Service
Discovery section
  scrape_interval: 30s               # how often to collect data
  scrape_timeout: 10s                # request timeout
  metrics_path: /metrics              # path to request
  scheme: http                       # http or https
  # Service discovery settings
  kubernetes_sd_configs:              # means that targets are obtained from Kubernetes
  - api_server: null                 # means to use the API server address from environment
variables (which are in every Pod)
    role: endpoints                  # take targets from endpoints
    namespaces:
      names:                        # search for endpoints only in these namespaces
      - foo
      - baz
  # "Filtering" settings (which endpoints to take and which not) and "relabeling" (which
labels to add or remove, on all received metrics)
  relabel_configs:
  # Filter by the value of the prometheus_custom_target label (obtained from the service
associated with the endpoint)
  - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
    regex: .+                       # any NON-empty label matches
    action: keep
  # Filter by port name
  - source_labels: [__meta_kubernetes_endpointslice_port_name]
    regex: http-metrics              # matches only if the port is called http-metrics
    action: keep
  # Add job label, use the value of the prometheus_custom_target label of the service, to
which we add the prefix "custom-"
  #
  # The job label is a Prometheus service label:
  #    * it determines the name of the group in which the target will be shown on the targets
page
  #    * and of course it will be on each metric obtained from these targets, so you can
easily filter in rules and dashboards
  - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
    regex: (.*)
    target_label: job
    replacement: custom-$1
    action: replace
  # Add namespace label
  - source_labels: [__meta_kubernetes_namespace]
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  # Add service label
  - source_labels: [__meta_kubernetes_service_name]
    regex: (.*)
    target_label: service
    replacement: $1
    action: replace
  # Add instance label (which will contain the Pod name)
  - source_labels: [__meta_kubernetes_pod_name]
    regex: (.*)
    target_label: instance
    replacement: $1
    action: replace
```

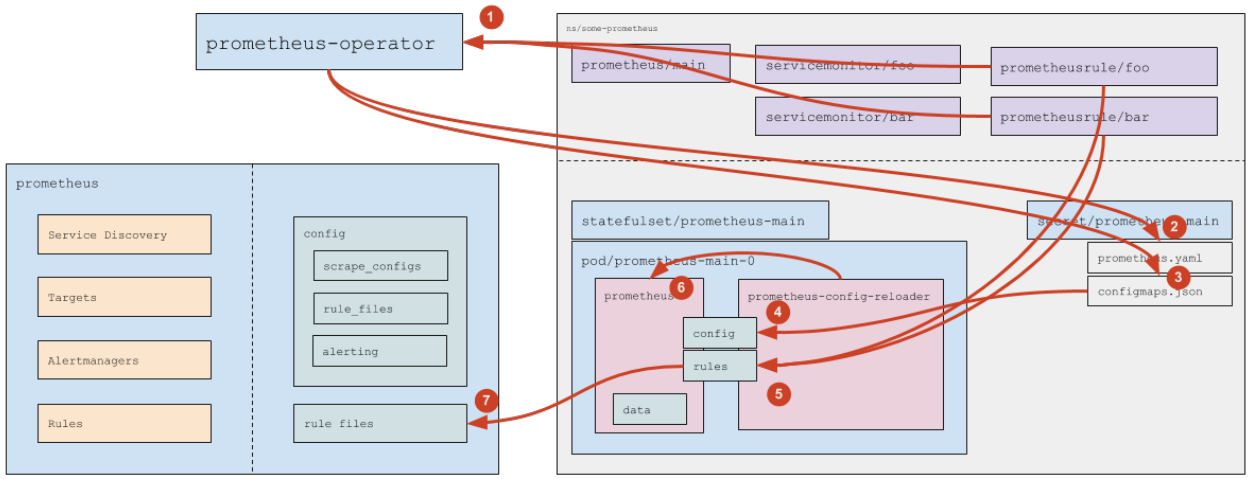
- Thus, Prometheus itself tracks:
  - Addition and removal of Pods (when Pods are added/removed, Kubernetes changes endpoints, and Prometheus sees this and adds/removes *targets*).
  - Addition and removal of services (more precisely endpoints) in specified namespaces.
- Config changes are required in the following cases:
  - Need to add a new scrape config (usually — a new type of services that need to be monitored).
  - Need to change the list of namespaces.

How are Service Monitors processed?



1. Prometheus Operator reads (and also monitors addition/removal/changes) Service Monitors (which specific Service Monitors — specified in the `prometheus` resource itself, see more details in official documentation).
2. For each Service Monitor, if it does NOT specify a specific list of namespaces (specified `any: true`), Prometheus Operator calculates (by accessing the Kubernetes API) a list of namespaces where there are Services (matching the labels specified in the Service Monitor).
3. Based on the read `servicemonitor` resources (see official documentation) and based on the calculated namespaces, Prometheus Operator generates part of the config (the `scrape_configs` section) and saves the config to the corresponding Secret.
4. Using standard Kubernetes means, data from the secret comes to the Pod (the `prometheus.yaml` file is updated).
5. The `prometheus-config-reloader` notices the file change and sends an HTTP request to Prometheus to reload.
6. Prometheus re-reads the config and sees changes in scrape_configs, which it processes according to its own logic (see more details above).

How are custom resources with *rules* processed?



1. Prometheus Operator monitors PrometheusRules (matching the `ruleSelector` specified in the `prometheus` resource).
2. If a new PrometheusRule appears (or an existing one is deleted) — Prometheus Operator updates `prometheus.yaml` (and then the logic exactly corresponding to Service Monitor processing, which is described above, is triggered).
3. Both in case of adding/removing PrometheusRule and when changing the content of PrometheusRule, Prometheus Operator updates the ConfigMap `prometheus-main-rulefiles-0`.
4. Using standard Kubernetes means, data from ConfigMap comes to the Pod
5. The `prometheus-config-reloader` notices the file change and:
   - Downloads changed ConfigMaps to the rules directory (this is `emptyDir`).
   - Sends an HTTP request to Prometheus to reload.
6. Prometheus re-reads the config and sees the changed *rules*.

Architecture of DP component availability assessment (upmeter)

Availability assessment in DP is performed by the upmeter module.

Composition of the upmeter module:

- **agent**: Works on master nodes and performs availability probes, sends results to the server.
- **upmeter**: Collects results and maintains an API server for their extraction.

- **front**:
  - **status**: Shows availability level for the last 10 minutes (requires authorization, but it can be disabled);
  - **webui**: Shows a dashboard with statistics on probes and availability groups (requires authorization).
- **smoke-mini**: Maintains continuous *smoke testing* using StatefulSet.

The module sends about 100 metric readings every 5 minutes. This value depends on the number of enabled Deckhouse Platform modules.

Additional resources

- Module `upmeter` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Prometheus module

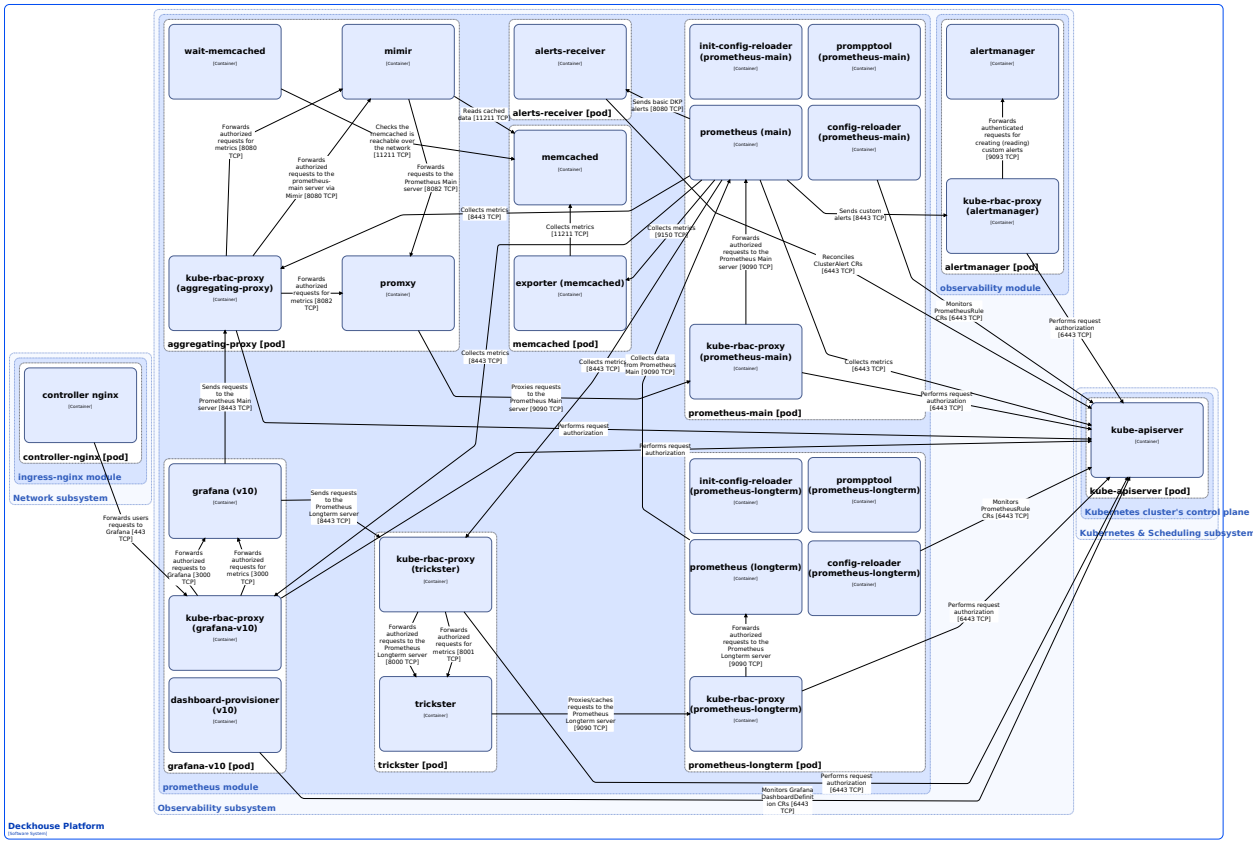
The `prometheus` module deploys a monitoring stack with predefined settings for Deckhouse Platform (DP) and provides the collection, storage, and processing of cluster and application metrics.

For more details about the module, refer to the module documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `prometheus` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Prometheus-main** (StatefulSet): Main Prometheus instance. Prometheus is a monitoring and notification system using a time series database (TSDB). It collects and analyzes application and server performance metrics in real

time. Prometheus-main collects metrics from configured monitoring objects every 30 seconds. You can use the `scrapeInterval` parameter to change this value.

Prometheus-main can use the original (“vanilla”) Prometheus or Deckhouse Prom++, a high-performance open-source fork of Prometheus designed to reduce memory consumption while maintaining full compatibility with the original project. The module uses Deckhouse Prom++ by default. It is possible to switch from Deckhouse Prom++ to the original Prometheus. In this case, migration of write-ahead log (WAL) data will be required, since Deckhouse Prom++ uses its own WAL log format. Migration is performed automatically using the prompptool init container.

Prometheus-main is the main data source. It collects metrics, processes configured rules, and sends alerts according to set configuration. The Prometheus Operator creates the Prometheus instance and its configuration based on following custom resources:

- Prometheus: Describes a Prometheus installation (cluster).
- ServiceMonitor: Defines metrics collection settings from a set of services.
- PrometheusRule: Contains a set of Prometheus rules.

Prometheus Operator watches Prometheus resources and, for each one, generates:

- StatefulSet with Prometheus itself.
- Secret containing `prometheus.yaml` (the main configuration file) and `configmaps.json` (the configuration file for the config-reloader container, described below). The `prometheus-main` Secret is mounted in prometheus-main Pod and is used by the config-reloader container.

Prometheus Operator also watches ServiceMonitor and PrometheusRule resources and, based on them, updates the configuration (`prometheus.yaml` and `configmaps.json`) in the described above Secret.

For more details about the working of Prometheus Operator, refer to the `operator-prometheus` module documentation section.

For more details about the prometheus-main operation, refer to the section describing the architecture of the monitoring in DP.

Prometheus-main consists of the following containers:

- **init-config-reloader**: Init container that performs a single run of config-reloader to load the Prometheus configuration.
- **prompptool**: Init container that performs automatic WAL data migration in case of switching from Deckhouse Prom++ to the original Prometheus and vice versa.
- **config-reloader**: Sidecar container that monitors changes in the `prometheus.yaml` configuration file and, if needed, triggers a Prometheus configuration reload (via a special HTTP request to the `/-/reload` endpoint). Config-reloader is a utility from the Prometheus Operator open-source project.
- **prometheus**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the Prometheus server. It is an open-source project.

2. **Prometheus-longterm** (StatefulSet): An additional Prometheus instance that scrapes the data of the primary Prometheus instance (`prometheus-main`). This allows users to view and analyze historical trends over a long period of time. Prometheus-longterm receives data due to a configured federation mechanism with the primary Prometheus instance.

   The original Prometheus or Deckhouse Prom++ can be used in prometheus-longterm too. Prometheus-longterm has the same set of containers as in prometheus-main, as well as operation principles.

3. **Grafana-v10**: Optional Grafana component that provides a web interface for visualizing monitoring data. Grafana displays dashboards supplied with DP modules. Grafana could work in High Availability mode, does not store state, and is configured with custom resources. Grafana is enabled by default, but it can be disabled using the `settings.grafana.enabled` parameter.

   > **i** Grafana-v10 will be disabled in the future DP releases. To view the monitoring dashboards, use the Deckhouse web UI.

   It consists of the following containers:

   - **dashboard-provisioner**: Sidecar container that watches GrafanaDashboardDefinition custom resources and when a new resource appears, it adds to the Grafana folder the dashboards described there.
   - **grafana**: Main container. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the Grafana server and its metrics (described above).

4. **Aggregating-proxy**: Component that performs metrics caching, data collection from several Prometheus instances (if they are in High Availability mode), data deduplication, and query calculation.

   It consists of the following containers:

   - **wait-memcached**: Init container that waits the memcached component to be available over the network. Aggregating-proxy uses memcached to cache metrics in RAM.
   - **mimir**: Sidecar container that works with the memcached component to optimize queries and cache data. If there is no data in the cache, mimir forwards the request to the prometheus-main component via promxy, that is another sidecar container. It is an open-source project.
   - **promxy**: Sidecar container that proxies requests to the prometheus-main component and provides a single endpoint to access data of multiple Prometheus instances. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the mimir containers (queries to the Prometheus server and container metrics) and promxy (queries to container metrics). It’s described above in detail.

5. **Memcached** (StatefulSet): Component used by aggregating-proxy for caching Prometheus metrics. Memcached is a software that implements a service for caching data in RAM to speed up Prometheus metrics query execution.

   It consists of the following containers:

   - **memcached**: Main container. It is an open-source project.
   - **exporter**: Sidecar container that exports memcached container metrics. Exporter collects memcached container metrics via a network connection, as well as from the memcached process's PID file. It is an open-source project.

6. **Trickster**: Caching proxy server that reduces the load on Prometheus. It is used for caching and proxying prometheus-longterm requests. In the future DP releases, it's expected to become deprecated and will no longer be supported.

   It consists of the following containers:

   - **trickster**: Main container. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the proxy server and its metrics (described above).

   > **i** In the future DP releases, alerts-receiver will be removed from the `prometheus` module. The Alertmanager component of the `observability` module will be used to receive all alerts.

7. **Alerts-receiver**: A server compatible with Alertmanager API. Alerts-receiver receives basic alerts from prometheus-main, creates the ClusterAlert custom resources based on them, updates their statuses and deletes them if the alert is no longer active. The ClusterAlert custom resources are used to inform DP users on active alerts and are displayed in the Deckhouse web UI. Alerts-receiver is developed by Flant. It consists of one container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches the PrometheusRule and GrafanaDashboardDefinition custom resources.
   - Manages the ClusterAlert custom resources.
   - Authorizes requests for metrics of module components.
2. **Alertmanager**: Sends custom alerts.

The Prometheus instance, which is part of the module, collects metrics from all DP components:

- Components of modules
- Components of cluster control plane
- Exporters that collect metrics of cluster hardware resources utilization
- Exporters that collect Kubernetes resources metrics
- User applications (additional configuration is required)

To prevent complicating the diagram with a large number of relations, Prometheus interactions related to the collection of metrics from DP components are omitted.

The following external components interact with the module:

1. **Ingress-controller** (shown on the diagram as "controller nginx"): Forwards users requests to Grafana.

Fault tolerance and High Availability (HA) monitoring mode

The `prometheus` module provides built-in fault tolerance for all its key components. All monitoring services (Prometheus servers, storage systems, proxies, and other important components) are deployed in multiple copies by default. This ensures that in the event of a failure of a separate instance, the service will continue to work without loss of data and availability.

Prometheus, the main component of metric collection, runs in at least two copies (if there are enough nodes in the cluster). All Prometheus instances use the same configuration and receive the same data. To ensure seamless operation in case of failure of one of the copies, a special component, the aggregating-proxy, is used to access Prometheus. It allows you to combine metrics from several Prometheus instances and always return the most complete and up-to-date data, even if one of the copies is temporarily unavailable.

Additional resources

- Module `console` documentation
- Module `observability` documentation
- Module `operator-prometheus` documentation
- Module `prometheus` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Observability module

The `observability` module extends the functionality of the `prometheus` module and the Deckhouse web UI, providing additional capabilities for flexible management of metrics, dashboards, and alerts, as well as access control mechanisms for them.

Module capabilities:

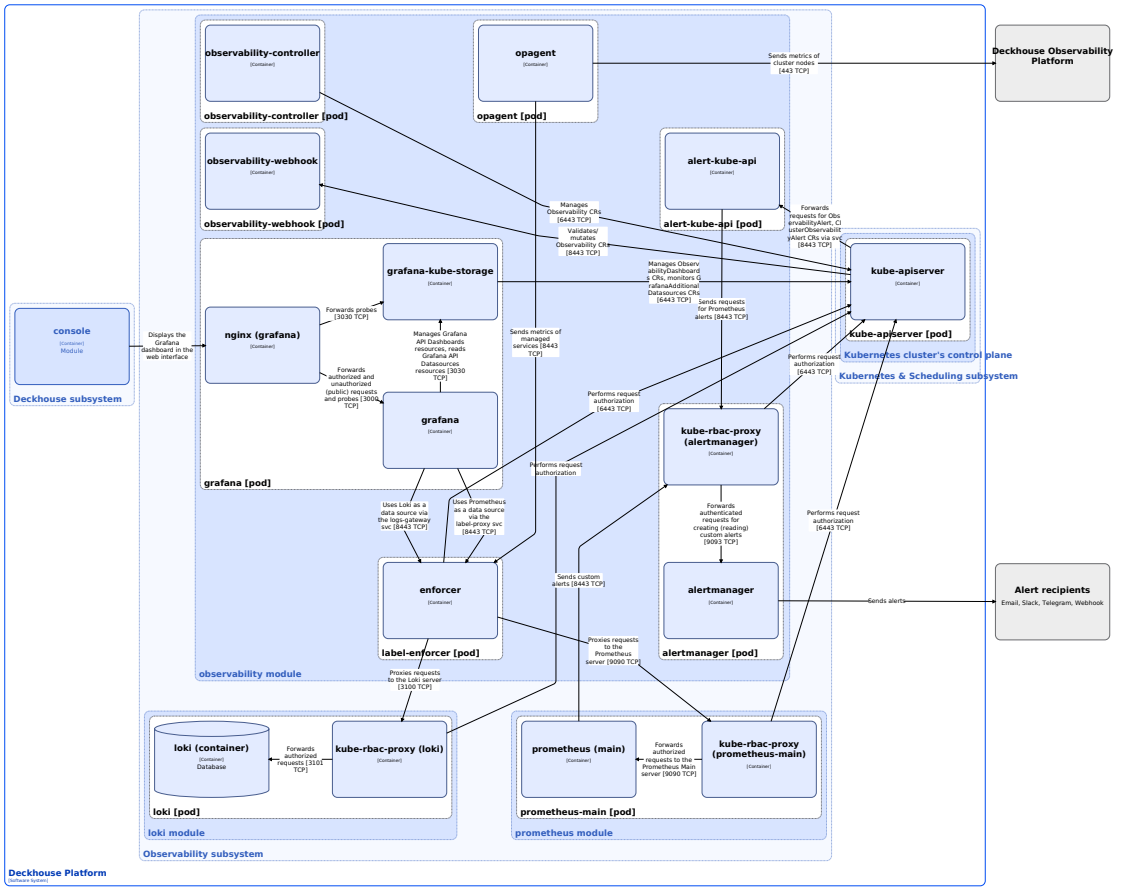
- **Dashboard management**: Allows users to add their own dashboards in Grafana format.
- **Alerting and recording rule group management**: Allows creating and configuring custom alerting and recording rule groups.
- **Notification configuration**: Allows configuring notification channels (Telegram, Slack, email, and webhooks), notification policies, and disabling notifications when needed.
- **Alert management**: Provides information about active alerts and stores the history of resolved alerts.
- **Standard data sources provisioning**.
- **Custom data sources support**: The module allows you to add custom data sources in addition to the provided set of standard data sources.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `observability` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Observability-controller**: Consists of a single **observability-controller** container and manages the lifecycle of most of the module's custom resources, such as ObservabilityMetricsRulesGroup, ObservabilityNotificationChannels, ObservabilityNotificationSilence, and others. For complete list of resources managed by the module, refer to the "Custom resources" section in the module documentation.

2. **Observability-webhook**: Consists of a single **observability-webhook** container that implements a webhook server, which is used to validate and mutate Kubernetes API resources through the Validating/Mutating Admission Controllers mechanisms.

3. **Alert-kube-api**: Consists of a single **alert-kube-api** container and implements Kubernetes Extension API Server that extends Kubernetes API with ObservabilityAlert и ClusterObservabilityAlert custom resources. Alert-kube-api

allows you to request alerts as custom resources using the Alertmanager component as a backend and caches them in memory for quick access.

4. **Alertmanager**: Receives alerts from prometheus-main component of the `prometheus` module, processes and sends them to the end recipients. DP supports sending alerts through the following delivery channels:

   - `Email`
   - `Telegram`
   - `Slack`
   - `Webhook`

   It consists of the following containers:

   - **alertmanager**: Main container. A fork of the original Alertmanager made by Flant that supports multitenancy, which allows dividing alerts by system (cluster) and project ones. The delivery of alerts (policies, channels, silencers) is also multitenant, allowing individual configuration of channels and delivery policies in different projects.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the Alertmanager API endpoint. It is an open-source project.

5. **Grafana**: A component that provides a web interface for visualizing monitoring data. A fork of Grafana made by Flant is used in the module. The modified Grafana has advanced features, such as separate access to metrics and dashboards according to multitenancy. Grafana dashboards of the `observability` module are integrated into the Deckhouse web UI (monitoring system management from one window).

   It consists of the following containers:

   - **grafana**: Main container.
   - **grafana-kube-storage**: Sidecar container that implements a backend for grafana container and providing management of Dashboard resources and reading Datasource resources of the Grafana component API. These resources allow you to view and manage dashboards within namespaces (projects), as well as to connect custom data sources.
   - **nginx**: Sidecar container that is an NGINX proxy server, which is used to publish static files. It is an open-source project.

6. **Label-enforcer**: A component that authorizes and proxies user requests to metrics (prometheus-main via the label-proxy service) and logs (loki via the logs-gateway service) sources, specified in the Datasource resources of grafana component API. Label-enforcer verifies RBAC access to monitoring data based on user rights, retrieves a list of available namespaces, and enriches requests with labels to filter requested data within user namespaces. For more information about metrics access control, refer to the module documentation. Label-enforcer handles both metrics read and write requests.

   It consists of a single container named **enforcer**.

7. **Opagent** (DaemonSet): An agent designed to collect metrics from both the operating system and the application software installed on the servers. OpAgent is developed by Flant for the Deckhouse Observability Platform (DOP) based on Okagent (Okmeter agent), that is also a part of the Okmeter monitoring system.

   In the `observability` module, opAgent connects to managed services, such as `managed-postgres`, `managed-memcached`, `managed-kafka`, and others (the list of supported managed services is constantly expanding), collects metrics from them, and sends them to prometheus-main component of the `prometheus` module. If the `observability-platform` module is enabled, opAgent collects metrics from the cluster nodes and sends them to DOP.

   OpAgent sends the collected data using the Prometheus Remote Write protocol:

   - Metrics from managed services: To Prometheus via label-enforcer.
   - Metrics from cluster nodes: To DOP.

   It consists of a single container named **opagent**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests for monitoring data.
   - Manages module custom resources.
2. **Prometheus**: Uses it as a data source and destination.
3. **Loki**: Uses it as a data source.
4. **Alerts receivers**: Sends alerts.
5. DOP: Uses it as a data destination (metrics from cluster nodes).

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates and mutates module custom resources (with validating and mutating webhooks).
   - Forwards to alert-kube-api requests for ObservabilityAlert and ClusterObservabilityAlert custom resources.
2. **Prometheus-main**: Sends alerts to alertmanager.
3. **Deckhouse web UI**: Uses Grafana for visualizing monitoring data.

Additional resources

- Module `console` documentation
- Module `managed-kafka` documentation
- Module `managed-memcached` documentation
- Module `managed-postgres` documentation
- Module `observability` documentation
- Module `prometheus` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Extended-monitoring module

The `extended-monitoring` module extends cluster monitoring capabilities with additional Prometheus exporters, which allow you to identify potential problems before they affect the operation of services.

Module features:

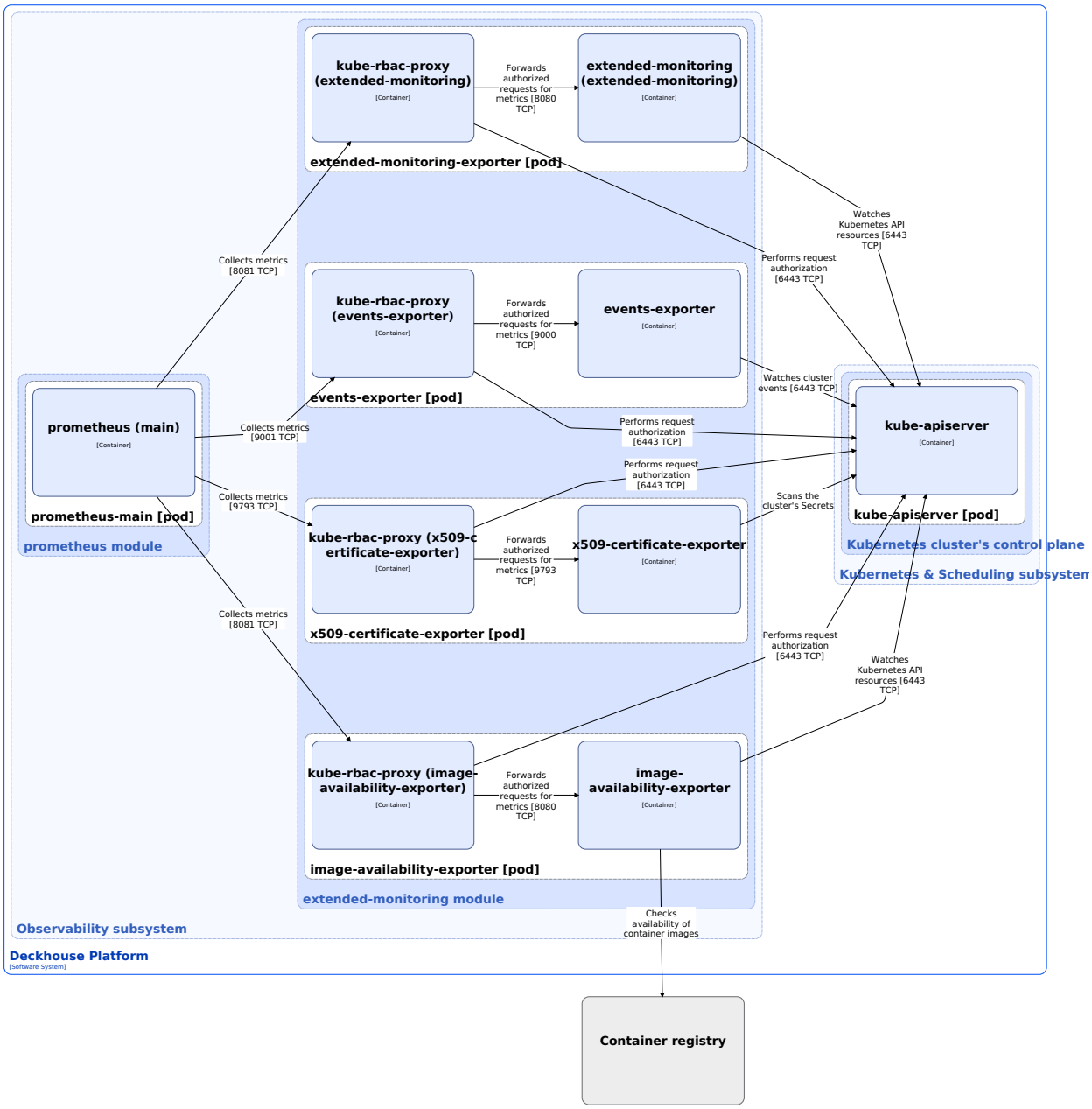
- Advanced metrics collection.
- Container image monitoring.
- Cluster events collection.
- Certificate control.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `extended-monitoring` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagrams.



Module components

The module consists of the following components:

1. **Extended-monitoring-exporter**: Prometheus exporter that collects and expounds metrics for free space and inodes on nodes, as well as for objects with a label `extended-monitoring.deckhouse.io/enabled =""` in the namespace. For more details about configuration of extended-monitoring metrics, refer to the corresponding documentation section. The module also contains Prometheus rules, based on which the Prometheus generates alerts. This allows you to detect and diagnose incidents faster.

   It consists of the following containers:

   - **extended-monitoring-exporter**: Main container. This exporter is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to exporter metrics. It is an open source project.

2. **Image-availability-exporter**: Prometheus exporter that monitors container images of pods running in cluster. Image-availability-exporter adds metrics about unavailability of container images to registry for all types of workload (Deployments, StatefulSets, DaemonSets, CronJobs). The module also contains appropriate Prometheus rules, based on which the Prometheus generates alerts. This helps to find out in advance about possible problems with launching or updating pods.

   It consists of the following containers:

   - **image-availability-exporter**: Main container. This exporter is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to exporter metrics (described above).

3. **Events-exporter**: Prometheus exporter that collects Kubernetes events and displays them as metrics, which allow you to track the dynamics of changes and respond faster to incidents.

   It consists of the following containers:

    - **events-exporter**: Main container. This exporter is developed by Flant.
    - **kube-rbac-proxy**: Sidecar container providing authorized access to exporter metrics (described above).

4. **X509-certificate-exporter**: Prometheus exporter that provides certificate control:

    - Scans the cluster's Secrets and generates metrics about the expiration of x509 certificates.
    - The module also contains appropriate Prometheus rules, based on which the Prometheus generates alerts. This allows you not to miss critical moments and update certificates on time, avoiding application downtime due to expired certificates.

    It consists of the following containers:

    - **x509-certificate-exporter**: Main container. It is an open source project;
    - **kube-rbac-proxy**: Sidecar container providing authorized access to exporter metrics (described above).

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Watches Kubernetes API resources.
    - Watches cluster events (Events resources).
    - Scans the cluster's Secrets.
    - Authorizes requests for metrics.

2. **Container registry**: Checks availability of container images.

The following external components interact with the module:

1. **Prometheus-main**: Collects metrics from module exporters.

Additional resources

- Module `extended-monitoring` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Monitoring-kubernetes module

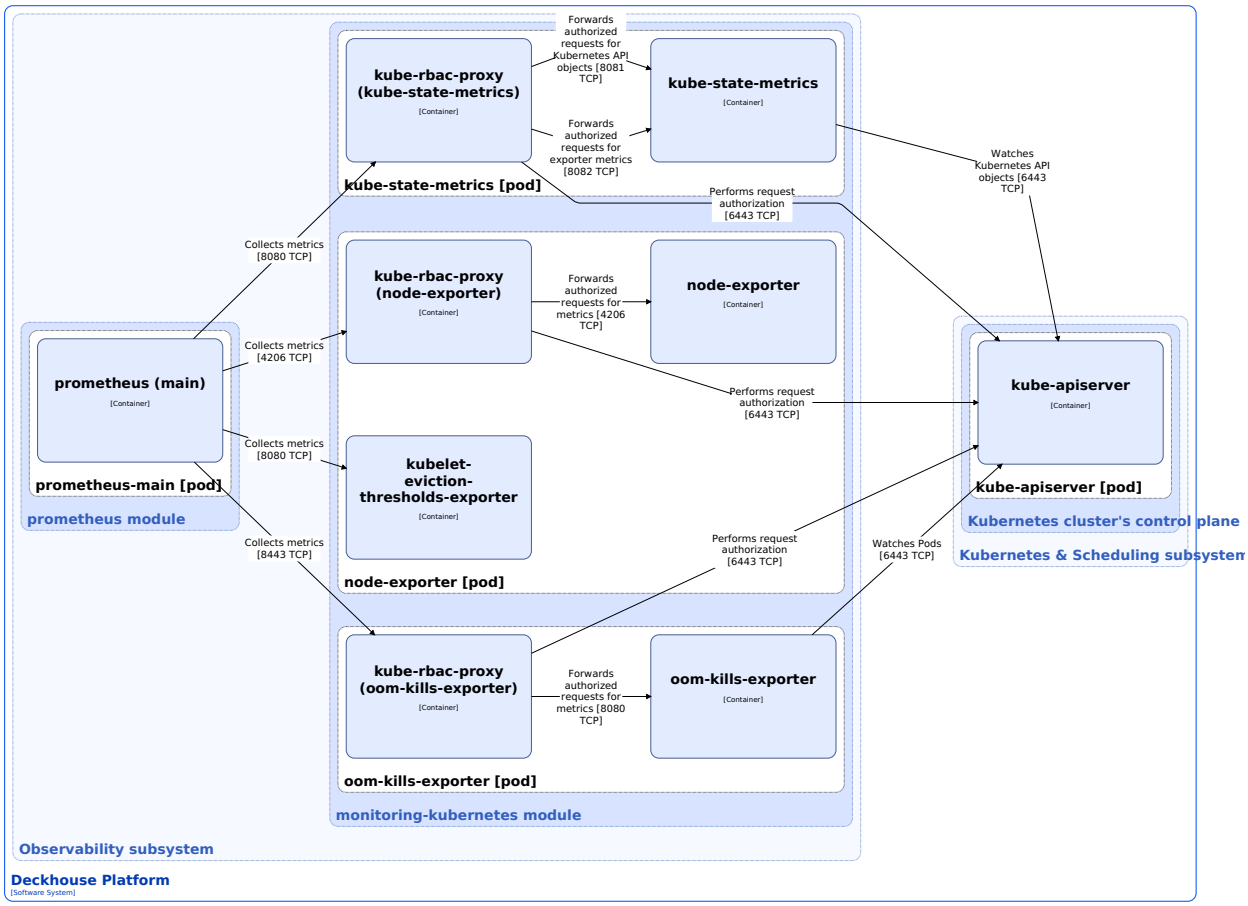
The `monitoring-kubernetes` module provides transparent and timely monitoring of the status of all cluster nodes and key infrastructure components.

For more details about module features and configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `monitoring-kubernetes` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Kube-state-metrics** (Deployment): Prometheus exporter that collects and provides metrics about the state of objects in a Kubernetes cluster. It connects to the Kubernetes API server, analyzes the current status of resources (Pods, Services, Deployments, Nodes, and others), and generates metrics based on this.

   It consists of the following containers:

   - **kube-state-metrics**: Main container. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the exporter metrics. It is an open-source project.

2. **Node-exporter** (DaemonSet): Prometheus exporter that runs on each cluster node and collects metrics from these nodes. It collects data about the operating system and hardware.

   It consists of the following containers:

   - **node-exporter**: Main container. It is an open-source project;
   - **kubelet-eviction-thresholds-exporter**: Sidecar container that collects metrics on the availability of file descriptors, sockets, free space and inodes on each node, compares them as a percentage with Eviction thresholds, stated in the kubelet configuration file, and exports the resulting values as metrics. This exporter is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the exporter metrics (described above).

3. **Oom-kills-exporter** (DaemonSet): Prometheus exporter that runs on each cluster node, monitors OOM-kill (Out-Of-Memory kill) events occurring with containers running on these nodes, and exports a counter of such events as a metric. Oom-kills-exporter also watches pods to synchronize labels containing container IDs that are added to the metrics.

   It consists of the following containers:

   - **oom-kills-exporter**: Main container. This exporter is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the exporter metrics (described above).

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches Kubernetes API objects;
   - Authorizes requests for metrics.

The following external components interact with the module:

1. **Prometheus-main**: Collects metrics from module exporters.

Additional resources

- Module `monitoring-kubernetes` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Upmeter module

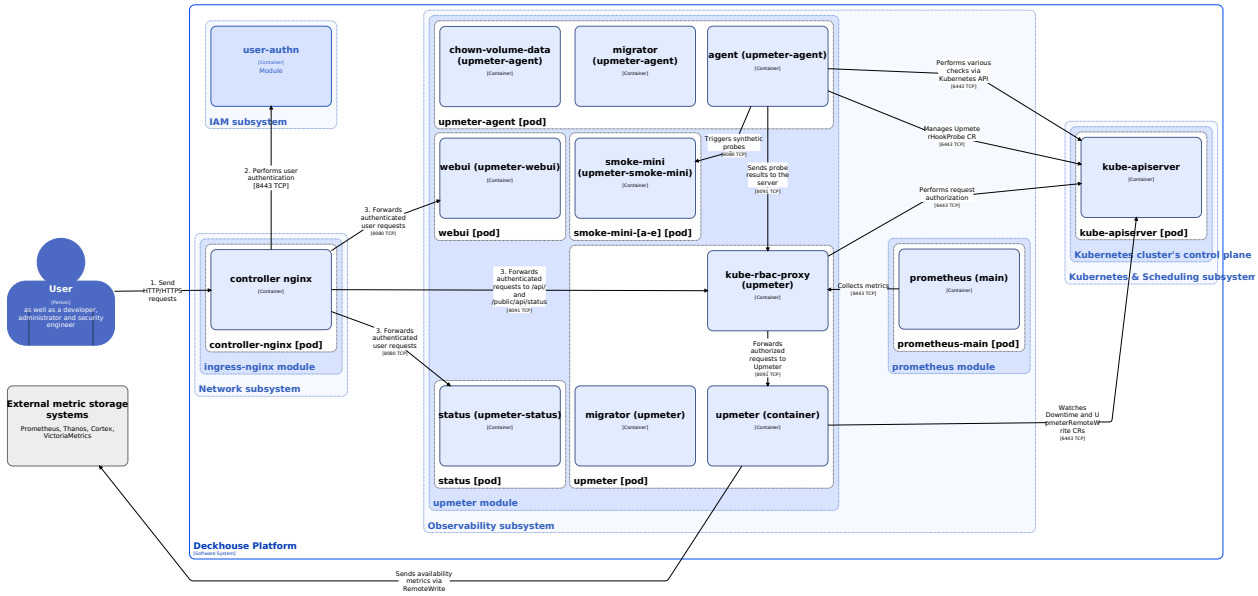
The `upmeter` module continuously checks Deckhouse Platform (DP) availability and cluster component health. Probe results are displayed on dashboards.

To learn more about module settings and usage examples, see the `upmeter` configuration page.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `upmeter` module and its interactions with other DP components are shown below.



Numbers in the diagram show the user request flow to the status and webui components. Steps 1, 2, and 3 handle requests through Ingress NGINX Controller with mandatory user authentication in the platform-wide auth system provided by the `user-authn` module. For details, see the `user-authn` architecture section.

Module components

The module includes the following components:

1. **Upmeter** (StatefulSet) is a controller that:
   - Watches the custom resource Downtime and calculates DP component availability excluding downtime intervals defined in this resource.
   - Stores DP component availability metrics in a local SQLite database.
   - Receives and processes DP component probe data.
   - Handles API requests for platform availability data.
   - Watches the custom resource UpmeterRemoteWrite and sends probe results to the endpoint defined in that resource by using the Prometheus Remote Write protocol.

   It includes the following containers:

   - **upmeter**: Main container.
   - **migrator**: Init container that applies SQL migrations to the component SQLite database.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC for secure access to the upmeter API.

2. **Upmeter-agent** (DaemonSet) runs on master nodes and regularly executes the following probe groups:

   - Control-plane: API server availability and controller health checks.
   - Deckhouse: DP cluster health and `deckhouse` module controller checks.
   - Extensions: Checks that every extension has at least one `Ready` Pod.
   - Load-balancing: Checks availability of network load balancing services.
   - Monitoring-and-autoscaling: Checks that the Observability subsystem is healthy and gathers metrics from system components.
   - Nginx: Checks that every Ingress NGINX Controller has at least one `Ready` Pod.
   - Nodegroups: Checks the number of `desired` nodes in each NodeGroup.
   - Synthetic: Checks network connectivity between cluster nodes with HTTP requests to smoke-mini-[a-e].

   The control-plane group includes the following probes:

   - Apiserver: Upmeter-agent checks Kubernetes API availability.
   - Basic-functionality: Upmeter-agent checks basic Kubernetes API behavior through the ConfigMap lifecycle.
   - Namespace: Upmeter-agent creates the `upmeter-probe-namespace` namespace and removes it after validation.
   - Scheduler: Upmeter-agent creates a Pod named `upmeter-probe-scheduler`, verifies that it is scheduled to any cluster node, and removes it.
   - Controller-manager: Upmeter-agent creates a StatefulSet named `upmeter-probe-controller-manager` with an intentionally missing container in the Pod spec, verifies that the target Pod is created and reaches the expected state, and then removes the StatefulSet.
   - Cert-manager: Upmeter-agent creates a self-signed Certificate named `upmeter-probe-cert-manager`, verifies that cert-manager created the related Secret, and then removes the Certificate and Secret.

   The following flow is used to check the deckhouse controller:

   - Upmeter-agent creates or updates the custom resource UpmeterHookProbe.
   - The deckhouse controller watches this resource and runs a hook to update it.
   - Upmeter-agent also watches UpmeterHookProbe and validates changes.

   You can disable probes or probe groups by using the `.spec.settings.disabledProbes` parameter.

   Upmeter-agent sends collected probe results to upmeter with an HTTP request `POST /downtime`.

   It includes the following containers:

   - **agent**: Main container.
   - **chown-volume-data**: Init container that sets required permissions for the `/var/lib/upmeter/agent` directory on a cluster node.
   - **migrator**: Init container that applies SQL migrations to the component SQLite database.

3. **Smoke-mini-[a-e]** (StatefulSet) is used for synthetic connectivity checks. It includes a single **smoke-mini** container. When upmeter-agent sends a request, smoke-mini-[a-e] instances send requests to each other and to the cluster DNS service, then return probe results.

   When the `upmeter` module is installed, the deckhouse controller from the `deckhouse` module registers a hook that distributes StatefulSet instances across different cluster nodes when possible. After that, it rebalances one StatefulSet every minute to another node.

4. **Status** (Deployment) includes a single **status** container and serves a web page with the current availability status of all DP components.

5. **Webui** (Deployment) includes a single **webui** container and serves a dashboard with per-component availability history.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Manages custom resources UpmeterRemoteWrite and UpmeterHookProbe.
   - Processes the Downtime custom resources.
   - Authorizes requests to upmeter.
   - Creates, validates, and deletes standard resources: Pod, StatefulSet, Namespace, and Secret, and the custom resource Certificate.

2. **External metric storage systems**: Sends probe results with the Prometheus Remote Write protocol.

The following external components interact with the module:

1. **Prometheus-main**: Uses monitoring rules and metrics related to the upmeter module.
2. **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `upmeter` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Logging modules

Log-shipper module

The `log-shipper` module simplifies log collection configuration in Kubernetes clusters. It allows collecting logs both from applications running in the cluster and from the cluster nodes, and then forwarding them to any log storage system — either internal or external (for example, Loki, Elasticsearch, and others).

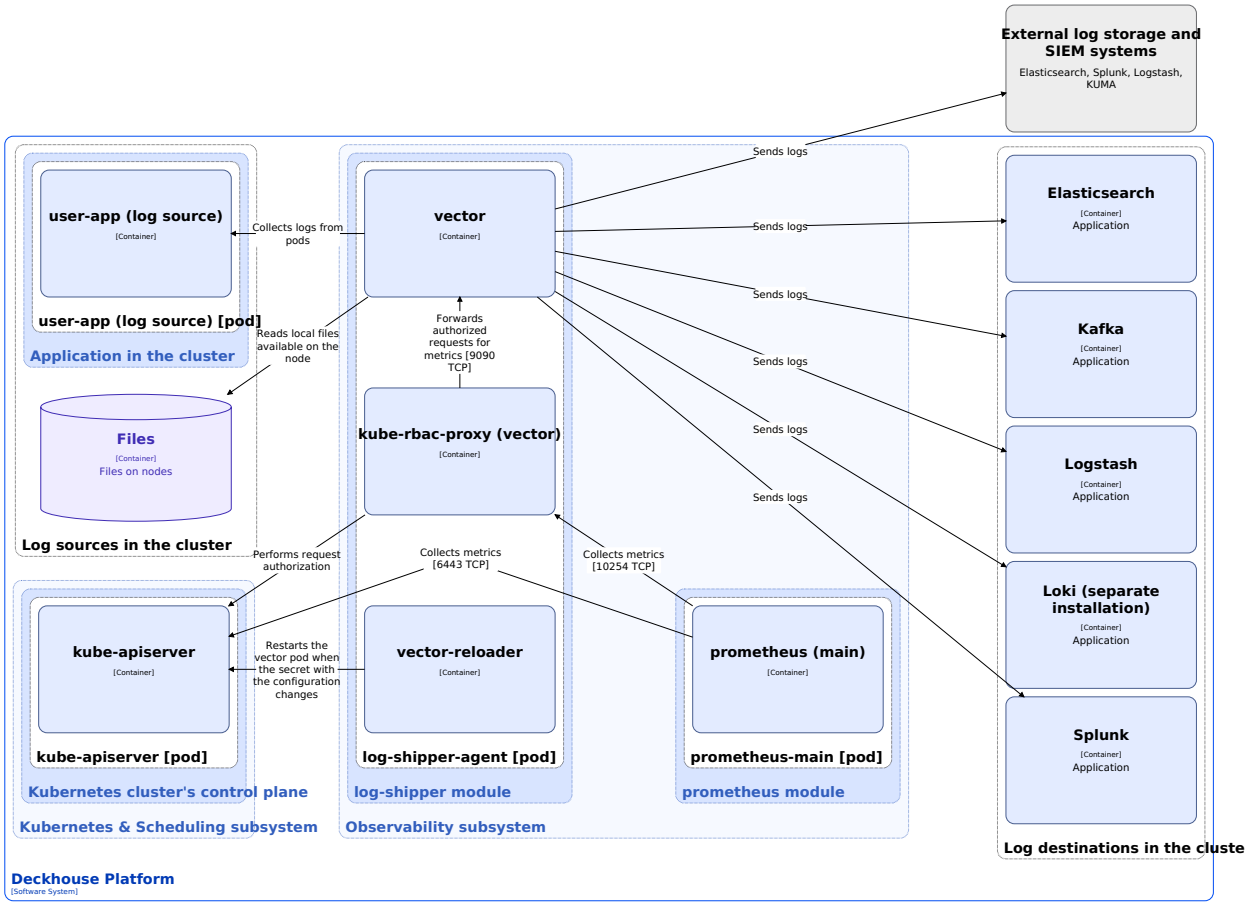
> **i** Deckhouse Platform (DP) provides integration with log storage systems. The storage systems themselves must be deployed and configured by the user.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `log-shipper` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of a single component:

- **log-shipper-agent** (DaemonSet): Separate instance of log-shipper-agent runs on each cluster node and includes the following containers:
  - **vector**: Logging agent based on Datadog Vector.

    It is configured using the custom resources ClusterLogDestination, ClusterLoggingConfig, and PodLoggingConfig.
  - **vector-reloader**: Sidecar container running Reloader. It monitors changes to the configuration Secret. When changes are detected, it validates the updated configuration and restarts vector to apply the updates.

- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to agent metrics.

Module interactions

The module interacts with the following components:

1. **Log sources in the cluster**:
   - Applications running in the cluster — the module collects logs from pods.
   - Files — the module reads local files on cluster nodes.
2. **Log destinations**:
   - Internal log storage systems.
   - External log storage systems and SIEM platforms.

   Both internal and external destinations may include Elasticsearch, Kafka, Logstash, Loki, and Splunk.
3. **Kube-apiserver**:
   - Authorizes requests for metrics.
   - Monitors changes to the Secret containing the vector configuration.

The following external components interact with the module:

1. **Prometheus-main** — collects metrics from log-shipper-agent.

Loki module

In Kubernetes, system logs stored on nodes are retained only for a limited time and may be lost during node restarts or updates. The `loki` module deploys an internal short-term log storage system in the cluster based on Grafana Loki.

Module capabilities:

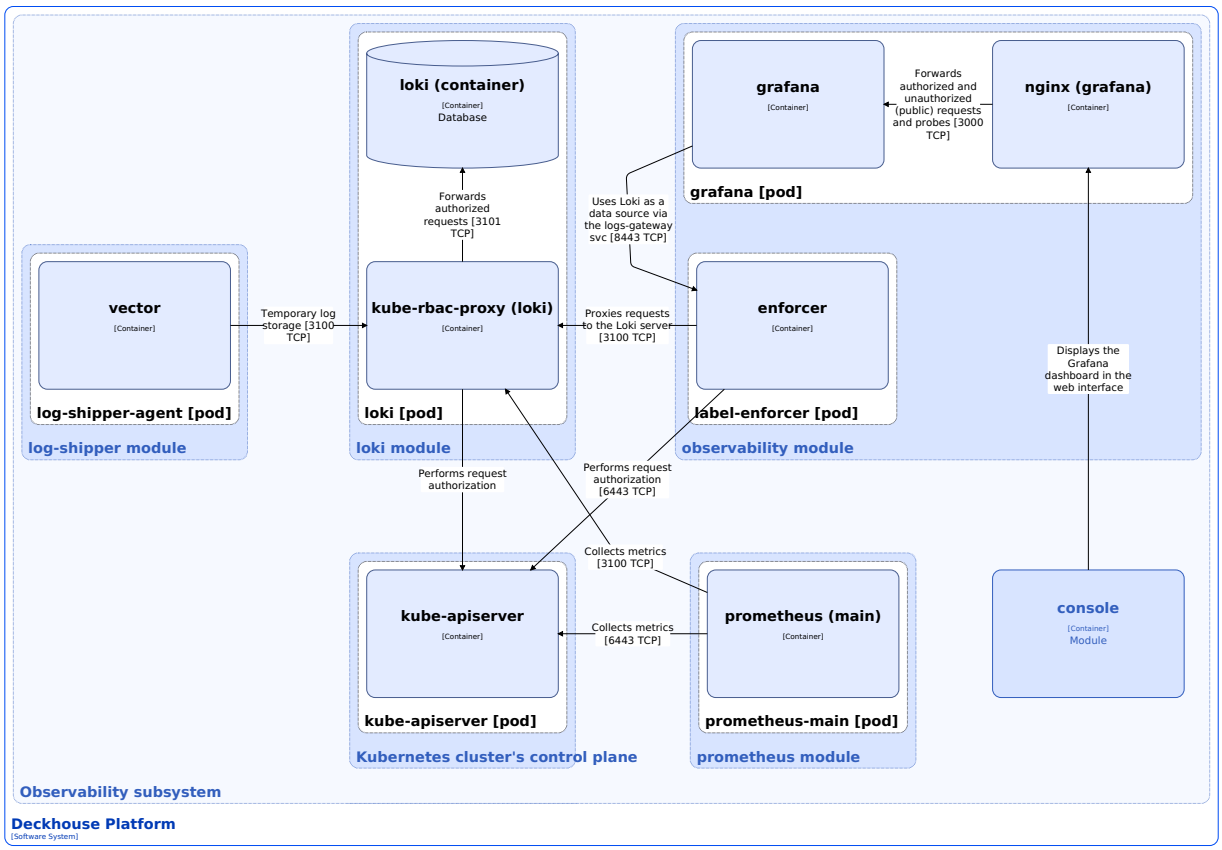
- System logs are automatically sent to Loki without additional configuration.
- Logs can be accessed through Grafana and the Deckhouse web UI (the `console` module).

> ⚠ The short-term log storage based on Grafana Loki does not support high availability (HA) mode. For long-term storage of important logs, use external storage systems supported by the `log-shipper` module.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

The Level 2 C4 architecture of the `loki` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of a single component:

- **loki**: *StatefulSet* with a single replica **loki-0**, which includes the following containers:
  - **loki**: Container running Grafana Loki.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to loki and its metrics.

Module interactions

The following external components interact with the module:

- **vector**: Sends logs from system components to loki.
- **console** (the Grafana instance embedded in the web UI): Uses loki as a data source for log visualization and analysis.
- **prometheus-main**: Collects loki metrics.

Additional resources

- Module `log-shipper` documentation
- Module `loki` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed Services subsystem

This subsection describes the architecture of the Managed Services subsystem of Deckhouse Platform (DP). Managed Services subsystem automates the deployment, scaling, backup, and updating of managed services in DP.

The Managed Services subsystem includes the following modules:

- `managed-cassandra`: Manages Cassandra instances.
- `managed-clickhouse`: Manages ClickHouse instances.
- `managed-hive-metastore`: Manages Hive Metastore instances.
- `managed-kafka`: Manages Kafka instances.
- `managed-memcached`: Manages Memcached instances.
- `managed-opensearch`: Manages OpenSearch instances.
- `managed-postgres`: Manages PostgreSQL clusters.
- `managed-starrocks`: Manages StarRocks instances.
- `managed-trino`: Manages Trino instances.
- `managed-valkey`: Manages Valkey instances.

Additional resources

- Module `managed-cassandra` documentation
- Module `managed-clickhouse` documentation
- Module `managed-hive-metastore` documentation
- Module `managed-kafka` documentation
- Module `managed-memcached` documentation
- Module `managed-opensearch` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-cassandra module

The `managed-cassandra` module manages instances of the distributed, Open Source NoSQL database system Apache Cassandra in Deckhouse Platform (DP). It provides:

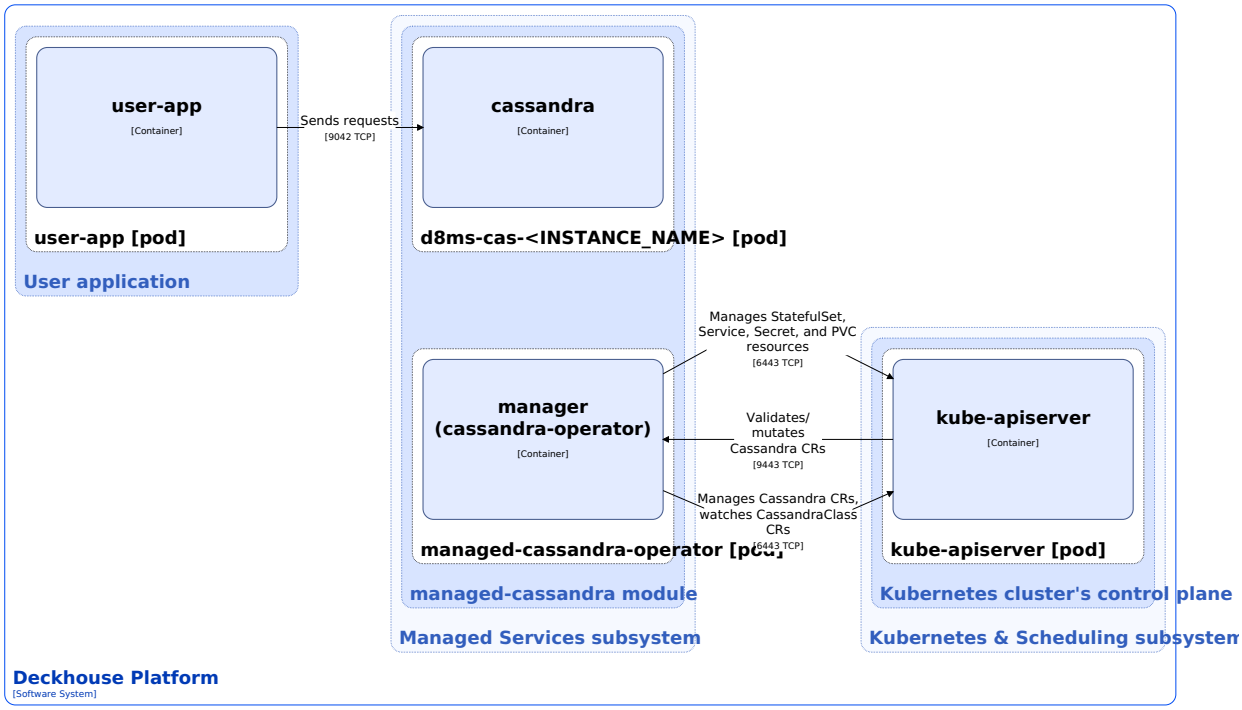
- **Automatic Deployment**: Creates a Cassandra instance using a simple YAML configuration.
- **Standalone**: Supports deployment of a standalone instance.
- **Configuration Management**: Separate CassandraClass resource for templating service creation and flexible validation of user parameters.
- **Status**: Displays the current status of the deployed Cassandra instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-cassandra` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-cassandra-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles Cassandra custom resources in all user namespaces. The Cassandra and CassandraClass custom resources define the settings of the Cassandra instance.
   - Creates and manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources related to the Cassandra instance.
   - Validates and mutates Cassandra custom resources using the Validating/Mutating Admission Controllers mechanism.
2. **D8ms-cas-<INSTANCE_NAME>** (StatefulSet): A component consisting of a single **cassandra** container that starts and runs the Cassandra instance. It is created by the managed-cassandra-operator component for each Cassandra custom resource.

Module interactions

The module interacts with the **kube-apiserver** component:

- Manages Cassandra custom resources.
- Watches CassandraClass custom resources.
- Manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Cassandra custom resources.
2. **User applications**: Send requests to the Cassandra instance.

Additional resources

- Module `managed-cassandra` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-clickhouse module

The `managed-clickhouse` module manages instances of ClickHouse in Deckhouse Platform (DP). ClickHouse is a high-performance Open Source columnar DBMS designed for online analytical processing (OLAP) of large volumes of data in real time. The module provides:

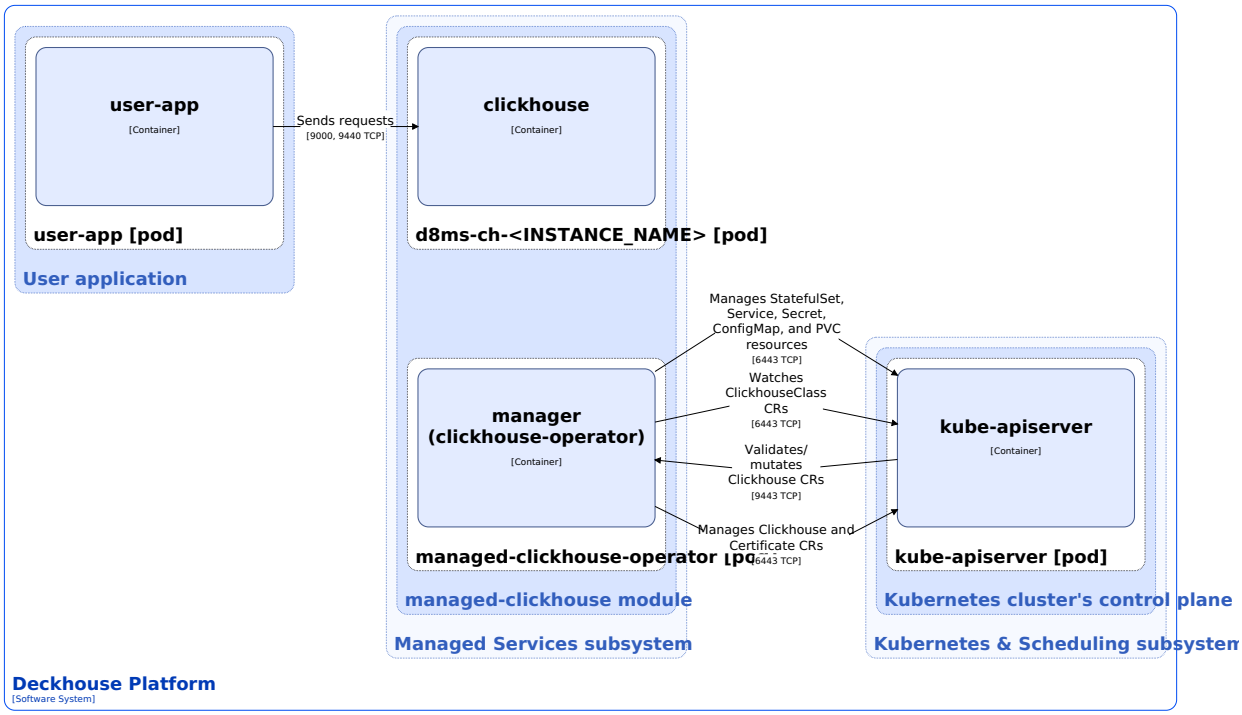
- **Automatic Deployment**: Creates a ClickHouse instance using a simple YAML configuration.
- **Standalone**: Supports deployment of a standalone instance.
- **Configuration Management**: Separate ClickhouseClass resource for templating service creation and flexible validation of user parameters.
- **Status**: Displays the current status of the deployed ClickHouse instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-clickhouse` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-clickhouse-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles Clickhouse custom resources in all user namespaces. The Clickhouse and ClickhouseClass custom resources define the settings of the ClickHouse instance.
   - Creates and manages the Certificate custom resource and StatefulSet, Secret, Service, ConfigMap, and PersistentVolumeClaim resources related to the ClickHouse instance.
   - Validates and mutates Clickhouse custom resources using the Validating/Mutating Admission Controllers mechanism.
2. **D8ms-ch-<INSTANCE_NAME>** (StatefulSet): A component consisting of a single **clickhouse** container that starts

and runs the ClickHouse instance. It is created by the managed-clickhouse-operator component for each Clickhouse custom resource.

Module interactions

The module interacts with the **kube-apiserver** component:

- Manages Clickhouse and Certificate custom resources.
- Watches ClickhouseClass custom resources.
- Manages StatefulSet, Secret, Service, ConfigMap, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Clickhouse custom resources.
2. **User applications**: Send requests to the ClickHouse instance.

Additional resources

- Module `managed-clickhouse` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-hive-metastore module

The `managed-hive-metastore` module manages instances of Hive Metastore (HMS) in Deckhouse Platform (DP). HMS is the centralized metadata store in the big data ecosystem. The module provides:

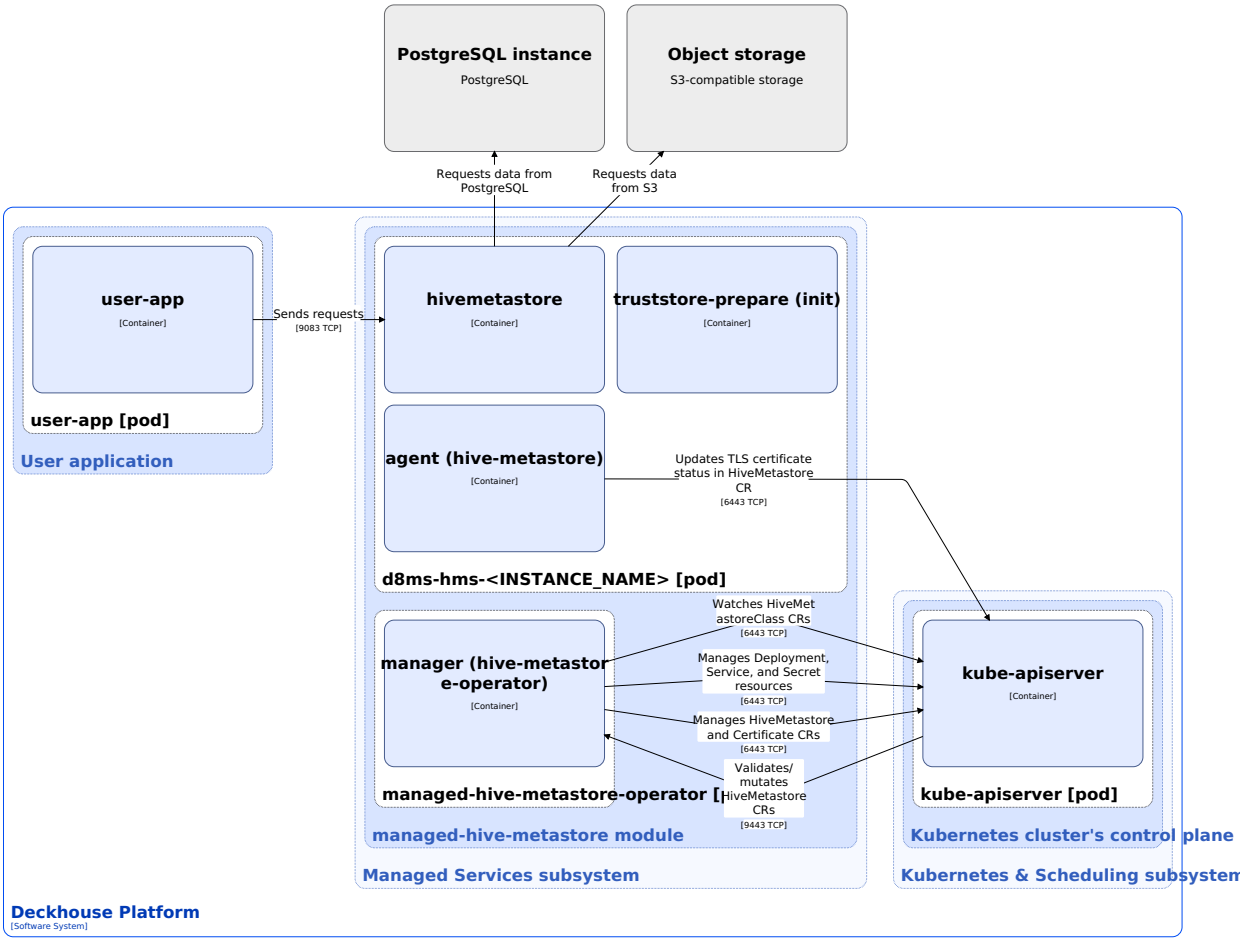
- **Automatic Deployment**: Creates an HMS instance using a simple YAML configuration.
- **Standalone**: Supports deployment of a standalone instance.
- **Configuration Management**: Separate HiveMetastoreClass resource for templating service creation and flexible validation of user parameters.
- **Status**: Displays the current status of the deployed HMS instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-hive-metastore` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-hive-metastore-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles HiveMetastore custom resources in all user namespaces. The HiveMetastore and HiveMetastoreClass custom resources define the settings of the HMS instance.
   - Creates and manages the Certificate custom resource and Deployment, Service, and Secret resources related to the HMS instance.
   - Validates and mutates HiveMetastore custom resources using the Validating/Mutating Admission Controllers mechanism.
2. **D8ms-hms-<INSTANCE_NAME>** (Deployment): A component that starts and runs the HMS instance.

   It consists of the following containers:
   - **truststore-prepare**: Init container that prepares the truststore.
   - **agent**: Sidecar container that monitors and updates the TLS certificate status.
   - **hivemetastore**: Main container.

Module interactions

The module interacts with the following components:

1. **PostgreSQL instance**: Processes metadata on the database server.
2. **Object storage**: Processes data in the object storage.
3. **Kube-apiserver**:
   - Manages HiveMetastore and Certificate custom resources.
   - Watches HiveMetastoreClass custom resources.
   - Manages Deployment, Service, and Secret resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates HiveMetastore custom resources.
2. **User applications**: Send requests to the HMS instance.

Additional resources

- Module `managed-hive-metastore` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-kafka module

The `managed-kafka` module manages Apache Kafka instances in Deckhouse Platform (DP). Apache Kafka is a distributed Open Source data streaming platform and message broker. The module provides:

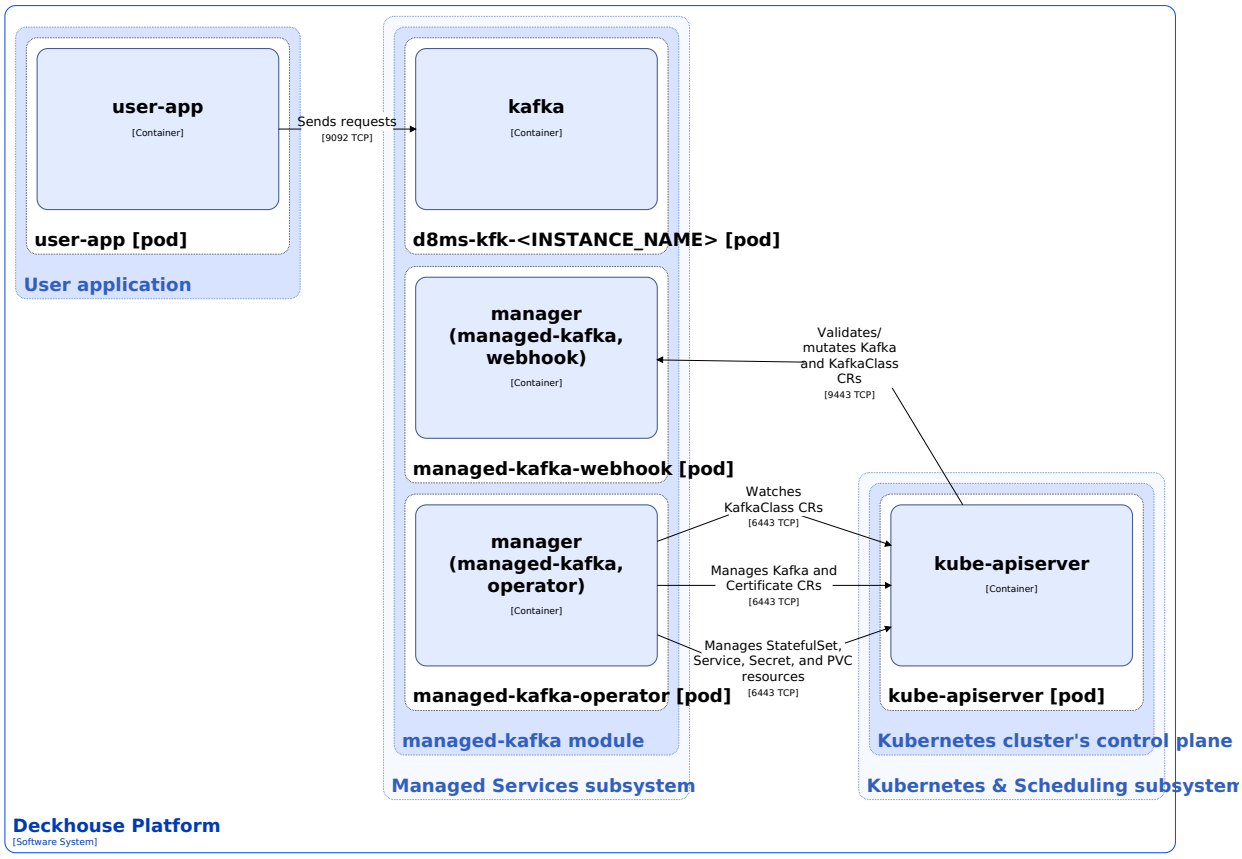
- **Automatic Deployment**: Creates a Kafka instance using a simple YAML configuration.
- **Configuration Management**: Separate KafkaClass resource for templating service creation and flexible validation of user parameters.
- **Long-term Storage**: Supports PersistentVolumeClaim parameters for data.
- **Status**: Displays the current status of the deployed Kafka instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-kafka` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-kafka-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles Kafka custom resources in all user namespaces. The Kafka and KafkaClass custom resources define the settings of the Kafka instance.

    - Creates and manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources related to the Kafka instance.

2. **Managed-kafka-webhook** (Deployment): A component consisting of a single container **manager**.

   It validates and mutates Kafka and KafkaClass custom resources using the Validating/Mutating Admission Controllers mechanism.

3. **D8ms-kfk-<INSTANCE_NAME>** (StatefulSet): A component consisting of a single **kafka** container that starts and runs the Kafka instance. It is created by the managed-kafka-operator component.

Module interactions

The module interacts with the **kube-apiserver** component:

- Manages Kafka and Certificate custom resources.
- Watches KafkaClass custom resources.
- Manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Kafka and KafkaClass custom resources.
2. **User applications**: Send requests to the Kafka instance.

Additional resources

- Module `managed-kafka` documentation

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-memcached module

The `managed-memcached` module simplifies the deployment and management of Memcached instances in Deckhouse Platform (DP). It provides:

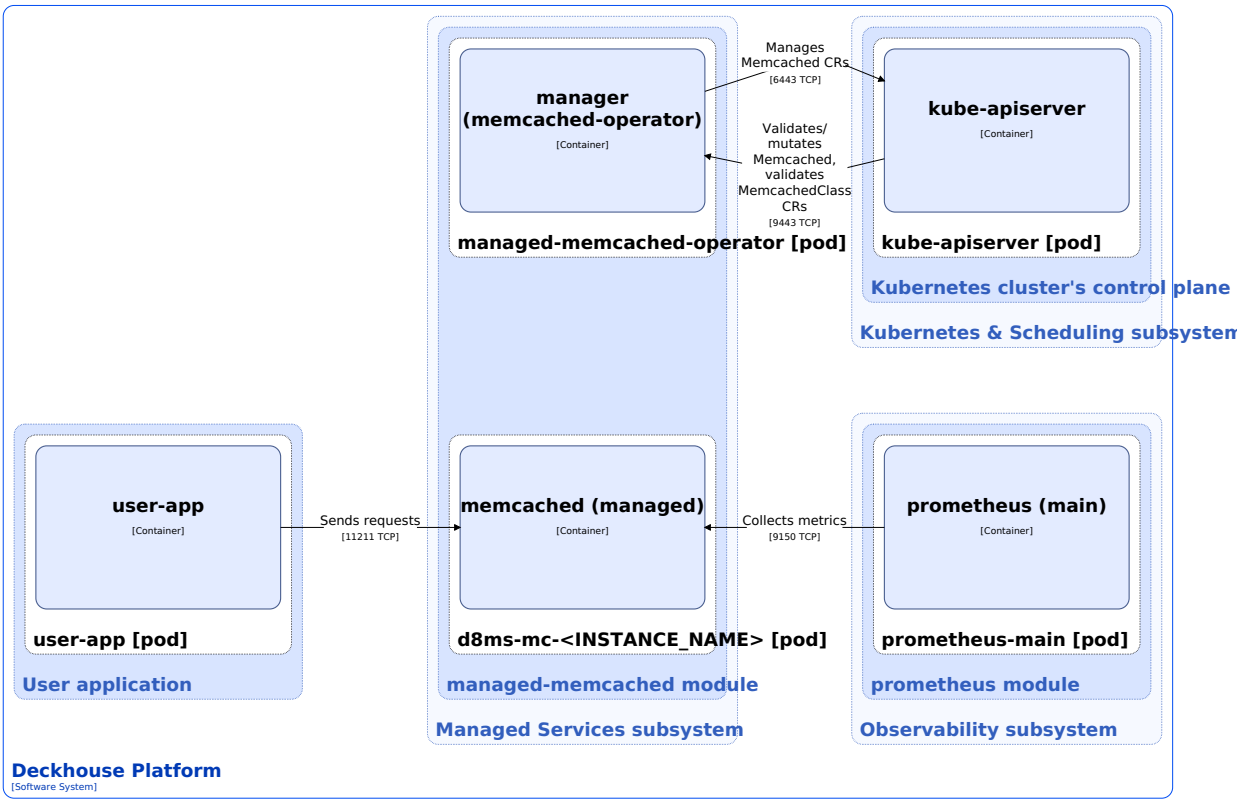
- **Automated Deployment**: Deploy Memcached instances with a simple YAML configuration.
- **High Availability**: Support for both standalone (Standalone) and group (Group) deployments.
- **Configuration Management**: Flexible configuration with validation and constraints through MemcachedClass.
- **Resource Management**: Automatic resource allocation and scaling.
- **Monitoring**: Built-in status tracking and health monitoring.
- **Security**: Distroless images and proper RBAC configuration.
- **Validation**: CEL rules for configuration validation.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-memcached` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-memcached-operator**: Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - It reconciles Memcached custom resources in all user namespaces. The Memcached resource defines the settings of the Memcached instance, including the topology and the deployment type.
   - It validates Memcached and MemcachedClass custom resources, and mutates Memcached custom resources using the Validating/Mutating Admission Controllers mechanism.
2. **d8ms-mc-<INSTANCE_NAME>** (StatefulSet): One or more Memcached instances, depending on the deployment type. They are created by the managed-memcached-operator component.

   It consists of a single container:
   - **memcached**: It is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**: Manages Memcached custom resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates Memcached and MemcachedClass custom resources, mutates Memcached custom resources.
2. **Prometheus-main**: Collects metrics from Memcached instances.
3. **User applications**: Sends requests to the Memcached instances.

Additional resources

- Module `managed-memcached` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-opensearch module

The `managed-opensearch` module manages instances of OpenSearch in Deckhouse Platform (DP). OpenSearch is an

Open Source search and analytics engine designed for working with large volumes of data in real time. The module provides:

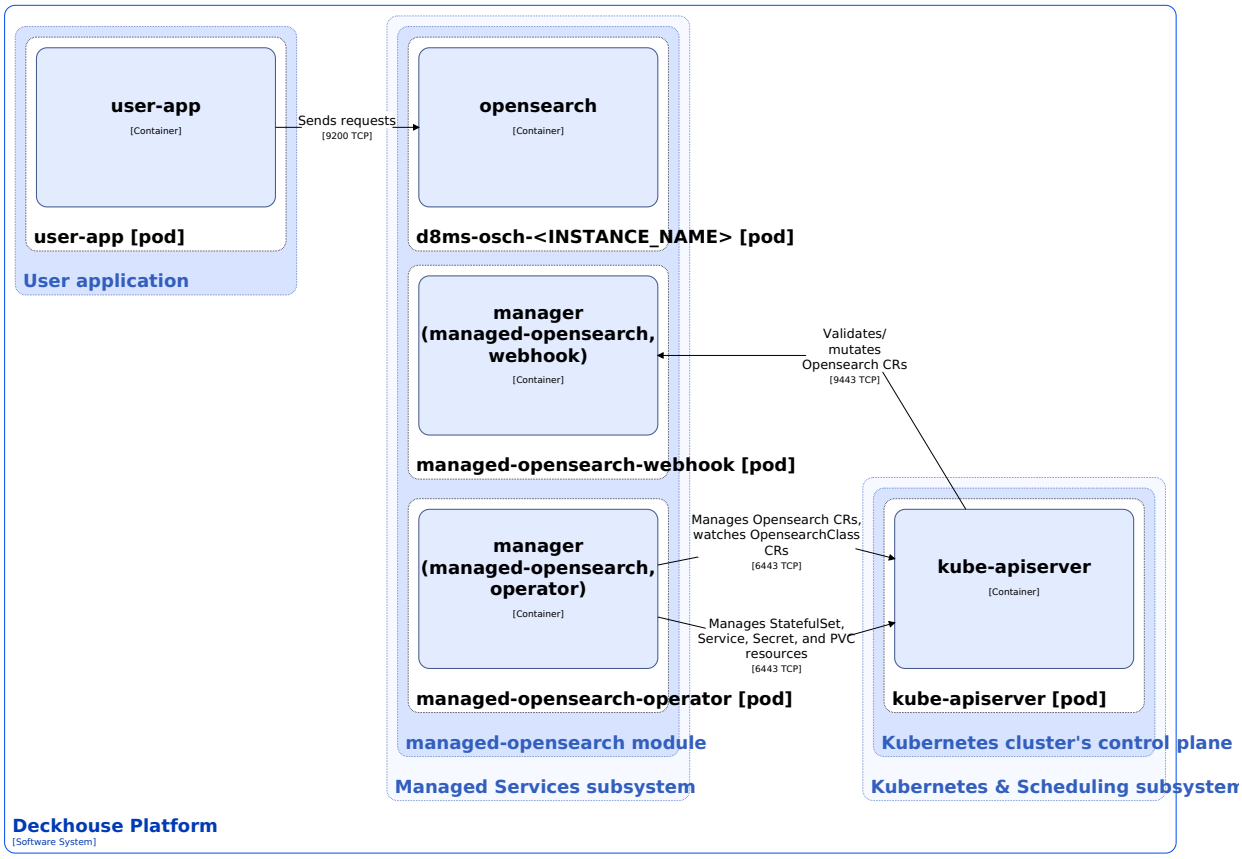
- **Automatic Deployment**: Creates an OpenSearch instance using a simple YAML configuration.
- **Configuration Management**: Separate OpensearchClass resource for templating service creation and flexible validation of user parameters.
- **Class-oriented Management**: Supports sizing policies and CEL validations.
- **Long-term Storage**: Supports PersistentVolumeClaim parameters for data.
- **Status**: Displays the current status of the deployed OpenSearch instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-opensearch` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-opensearch-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles Opensearch custom resources in all user namespaces. The Opensearch and OpensearchClass custom resources define the settings of the OpenSearch instance.
   - Creates and manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources related to the OpenSearch instance.
2. **Managed-opensearch-webhook** (Deployment): A component consisting of a single container **manager**.

   It validates and mutates Opensearch custom resources using the Validating/Mutating Admission Controllers mechanism.
3. **D8ms-osch-<INSTANCE_NAME>** (StatefulSet): A component consisting of a single **opensearch** container that starts and runs the OpenSearch instance. It is created by the managed-opensearch-operator component.

Module interactions

The module interacts with the **kube-apiserver** component:

- Manages Opensearch and Certificate custom resources.
- Watches OpensearchClass custom resources.
- Manages StatefulSet, Service, Secret, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Opensearch custom resources.
2. **User applications**: Send requests to the OpenSearch instance.

Additional resources

- Module `managed-opensearch` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-postgres module

The `managed-postgres` module manages PostgreSQL clusters in Deckhouse Platform (DP). It allows users to configure and scale PostgreSQL clusters according to their needs, ensuring optimal performance and security.

Main features:

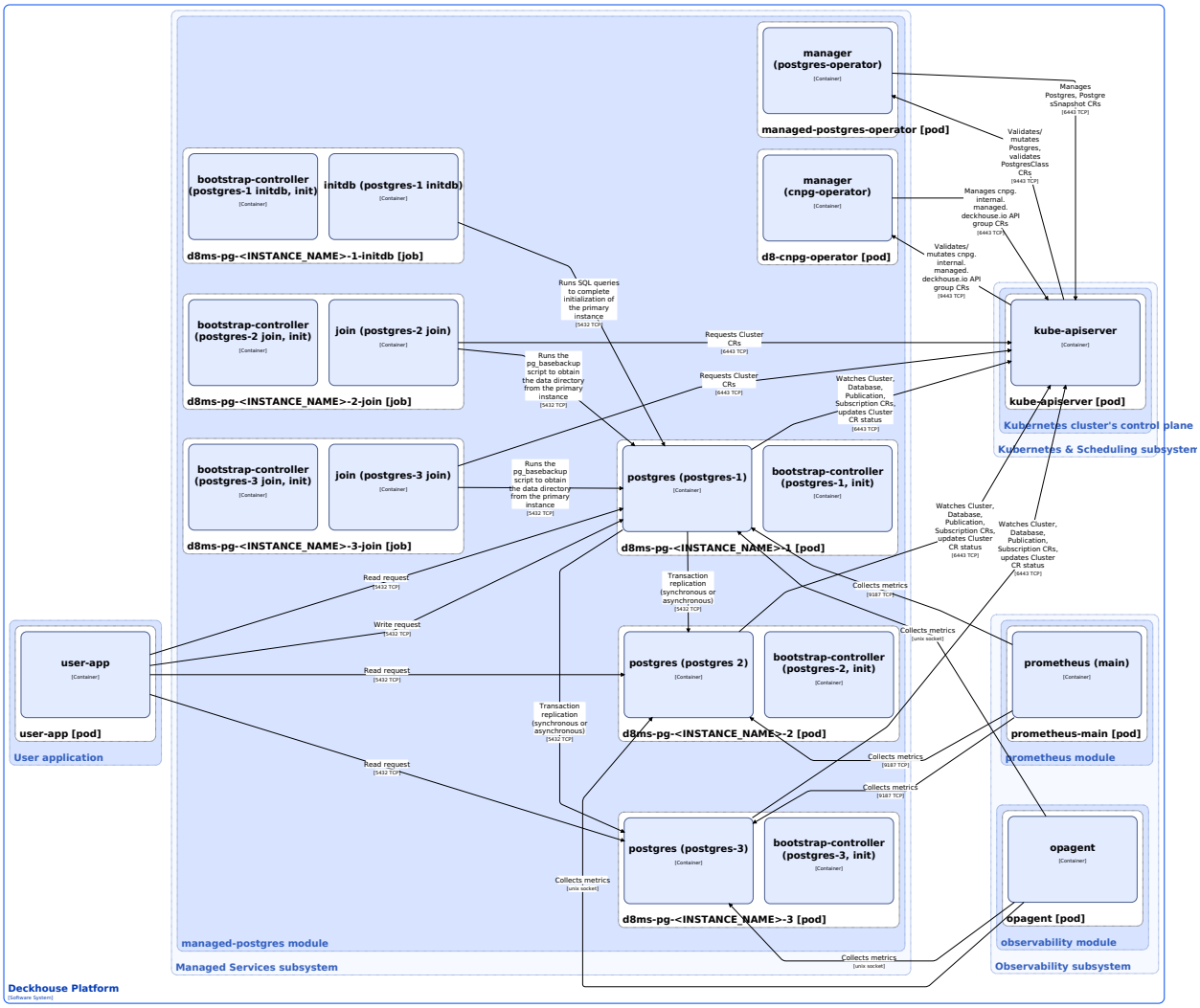
- **Automatic Deployment**: Deploys a Postgres instance using a simple YAML configuration.
- **High Availability**: Supports deployment of a highly available Postgres cluster or a standalone instance of your choice.
- **Configuration Management**: Separate PostgresClass custom resource for templating the cluster creation approach with the ability to flexibly validate user configs.
- **User and Database Management**: Declarative model for creating users and logical databases.
- **Status**: Informative set of states for tracking the deployed Postgres.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-postgres` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-postgres-operator**: Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - It reconciles Postgres custom resources in all user namespaces. The Postgres resource defines the settings of the PostgreSQL cluster, including the topology and the replication mode, PostgreSQL instances configuration, and other parameters such as lists of logical databases and internal users.
   - It reconciles PostgresSnapshot custom resources. PostgresSnapshot resource designed for PostgreSQL instances backup and restore;
   - It performs Postgres and PostgresClass custom resources validation, as well as Postgres custom resources mutation using the Validating/Mutating Admission Controllers mechanism.

   Managed-postgres-operator uses the `cnpg.internal.managed.deckhouse.io` API group custom resources managed by the d8-cnpg-operator component as a backend.
2. **D8-cnpg-operator**: A fork of CloudNativePG, a Kubernetes operator that automates the management of PostgreSQL clusters. D8-cnpg-operator consists of a single container **manager** and performs the following operations:
   - It manages the following `cnpg.internal.managed.deckhouse.io` API group custom resources:
     - Cluster: Defines a PostgreSQL cluster.
     - Pooler: Defines connection pool settings.
     - FailoverQuorum: It is used to display a quorum status of PostgreSQL cluster replicas in high-availability configuration.
     - Database: Defines a logical database.
     - Backup: Defines a PostgreSQL instance backup.
     - ScheduledBackup: Defines a PostgreSQL scheduled backup settings.
     - Subscription: Defines a logical replication destination.
     - Publication: Defines a logical replication source.
     - ImageCatalog: Defines PostgreSQL images for user namespaces.
     - ClusterImageCatalog: Defines cluster-wide PostgreSQL images.
   - It performs `cnpg.internal.managed.deckhouse.io` API group custom resources validation and mutation using the Validating/Mutating Admission Controllers mechanism.

> **i** The components for ConsistencyAndAvailability replication mode are described below. The components for other two replication modes are the part of components for ConsistencyAndAvailability mode.

3. **d8ms-pg-<INSTANCE_NAME>-1-initdb** (Job): A job created by the d8-cnpg-operator component that runs SQL queries to complete initialization of the primary PostgreSQL instance.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **initdb**: Main container that runs `manager` executable file, which executes the SQL queries described above.

4. **d8ms-pg-<INSTANCE_NAME>-2-join** (Job): A job created by the d8-cnpg-operator component that runs `pg_basebackup` script to get `data` directory from primary instance for first replica. This job uses a Cluster custom resource to configure the replica.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **join**: Main container that runs `manager` executable file, which executes the script described above.

5. **d8ms-pg-<INSTANCE_NAME>-3-join** (Job): A job created by the d8-cnpg-operator component that runs `pg_basebackup` script to get `data` directory from primary instance for second replica. This job uses a Cluster custom resource to configure the replica.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **join**: Main container that runs `manager` executable file, which executes the script described above.

6. **d8ms-pg-<INSTANCE_NAME>-1**: PostgreSQL primary instance. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs the following operations:
     - It starts PostgreSQL processes.
     - It manages PostgreSQL instance lifecycle, including server monitoring, handling server shutdown and restart.
     - It participates in `switchover`/`failover` procedures. `Switchover` is a planned and controlled process in which an active primary instance is intentionally decommissioned and an assigned backup instance (replica) is promoted to the primary instance role. Its main goal is to ensure zero data loss: before transferring the role to the replica, the system waits until all current transactions are replicated. A `failover` is an emergency situation: the primary instance has failed, become unavailable, or cannot be safely used to write data. In this case, the designated backup instance (replica) is promoted to the primary instance role, and data loss is possible.
     - It interacts with the operator and publishes instance status;
     - It watches Cluster, Database, Publication and Subscription custom resources.

7. **d8ms-pg-<INSTANCE_NAME>-2**: PostgreSQL first replica. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs operations described for PostgreSQL primary instance.

8. **d8ms-pg-<INSTANCE_NAME>-3**: PostgreSQL second replica. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs operations described for PostgreSQL primary instance.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages Postgres and PostgresSnapshot custom resources.
   - Manages `cnpg.internal.managed.deckhouse.io` API group custom resources.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates Postgres and PostgresClass custom resources, mutates Postgres custom resources.
   - Validates and mutates `cnpg.internal.managed.deckhouse.io` API group custom resources.
2. **Prometheus-main**: Collects PostgreSQL cluster instances metrics (which are exported by `manager` executable file of d8-cnpg-operator component that is run in postgres container of PostgreSQL instance pod).
3. **opAgent**: Collects PostgreSQL cluster instances metrics (connecting directly to PostgreSQL cluster instances) and sends them to prometheus-main.

4. **User applications**: Sends requests to PostgreSQL cluster instances. Write requests are sent to the primary instance via the **d8ms-pg-<INSTANCE_NAME>-rw** service. The primary instance replicates transactions to replicas. Read requests are balanced to PostgreSQL instances via **d8ms-pg-<INSTANCE_NAME>-r** or **d8ms-pg-<INSTANCE_NAME>-ro** services.

Additional resources

- Module `managed-postgres` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-starrocks module

The `managed-starrocks` module manages StarRocks instances in Deckhouse Platform (DP). StarRocks is a high-performance analytical DBMS (OLAP) for real-time analytics, data warehouses, and BI workloads.

The module provides:

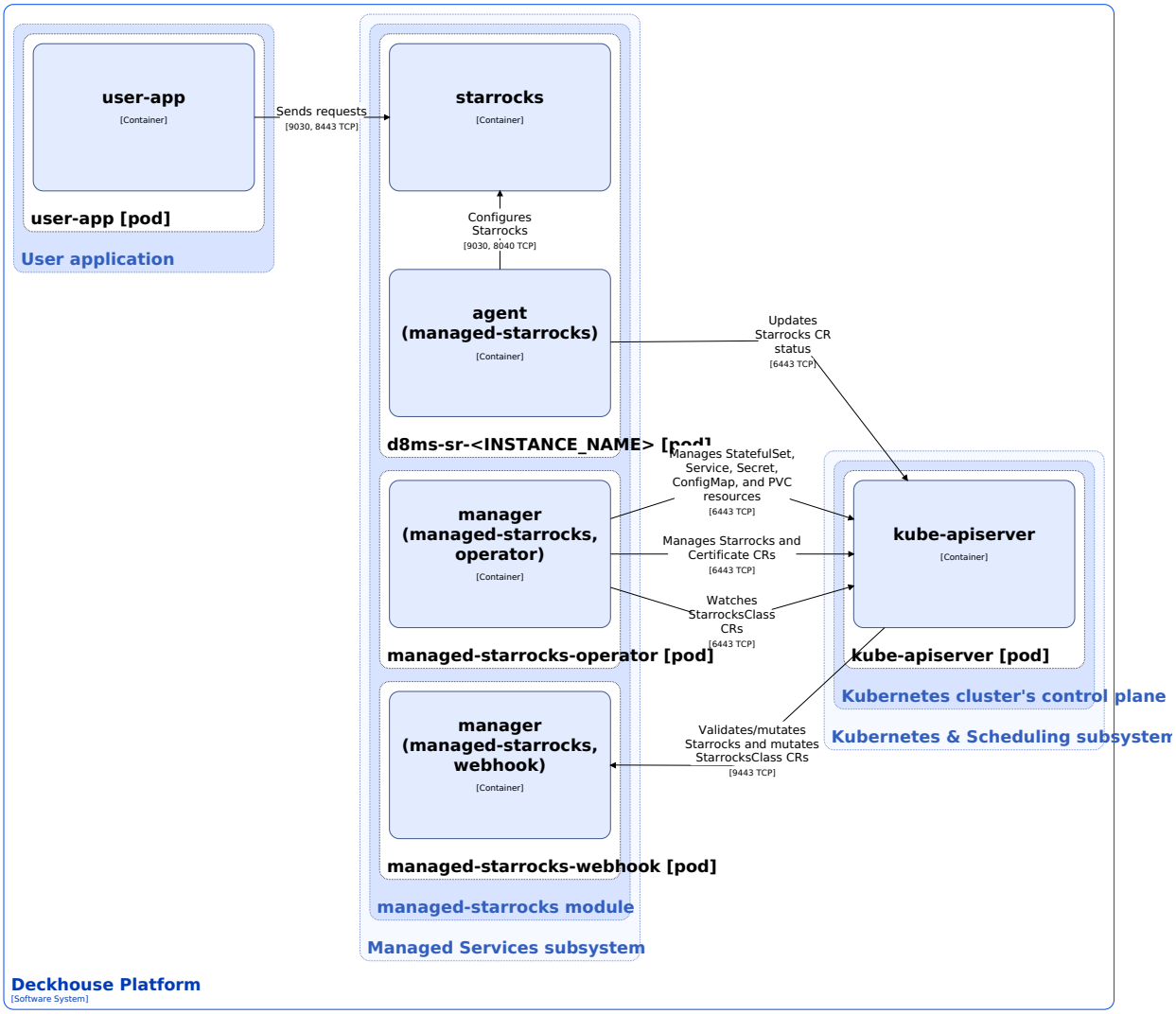
- **Automatic Deployment**: Creates a StarRocks instance using a simple YAML configuration.
- **Configuration Management**: Separate StarrocksClass resource for templating service creation and flexible validation of user parameters.
- **Analytics Tuning**: Supports data load parameters (`streamingLoadMaxMb`, `loadProcessMaxMemoryLimitPercent`) and catalog management (`catalogTrashExpireSecond`).
- **Status**: Displays the current status of the deployed StarRocks instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-starrocks` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-starrocks-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - Reconciles Starrocks custom resources in all user namespaces. The Starrocks and StarrocksClass custom resources define the settings of the StarRocks instance.
   - Creates and manages the Certificate custom resource and StatefulSet, Service, Secret, ConfigMap, and PersistentVolumeClaim resources related to the StarRocks instance.
2. **Managed-starrocks-webhook** (Deployment): A component consisting of a single container **manager**.

   It validates and mutates Starrocks custom resources, and mutates StarrocksClass custom resources using the Validating/Mutating Admission Controllers mechanism.
3. **D8ms-sr-<INSTANCE_NAME>** (StatefulSet): A component that starts and runs the StarRocks instance. It is created by the managed-starrocks-operator component.

   It consists of the following containers:
   - **agent**: Sidecar container that monitors and updates the TLS certificate status, and configures the instance.
   - **starrocks**: Main container.

Module interactions

The module interacts with the **kube-apiserver** component:

- Manages Starrocks and Certificate custom resources.
- Watches StarrocksClass custom resources.
- Manages StatefulSet, Service, Secret, ConfigMap, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Starrocks custom resources, mutates StarrocksClass custom resources.
2. **User applications**: Send requests to the StarRocks instance.

Additional resources

- Module `managed-starrocks` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-trino module

The `managed-trino` module manages Trino instances in Deckhouse Platform (DP). Trino is a fast Open Source distributed SQL engine built for big data analytics. The module provides:

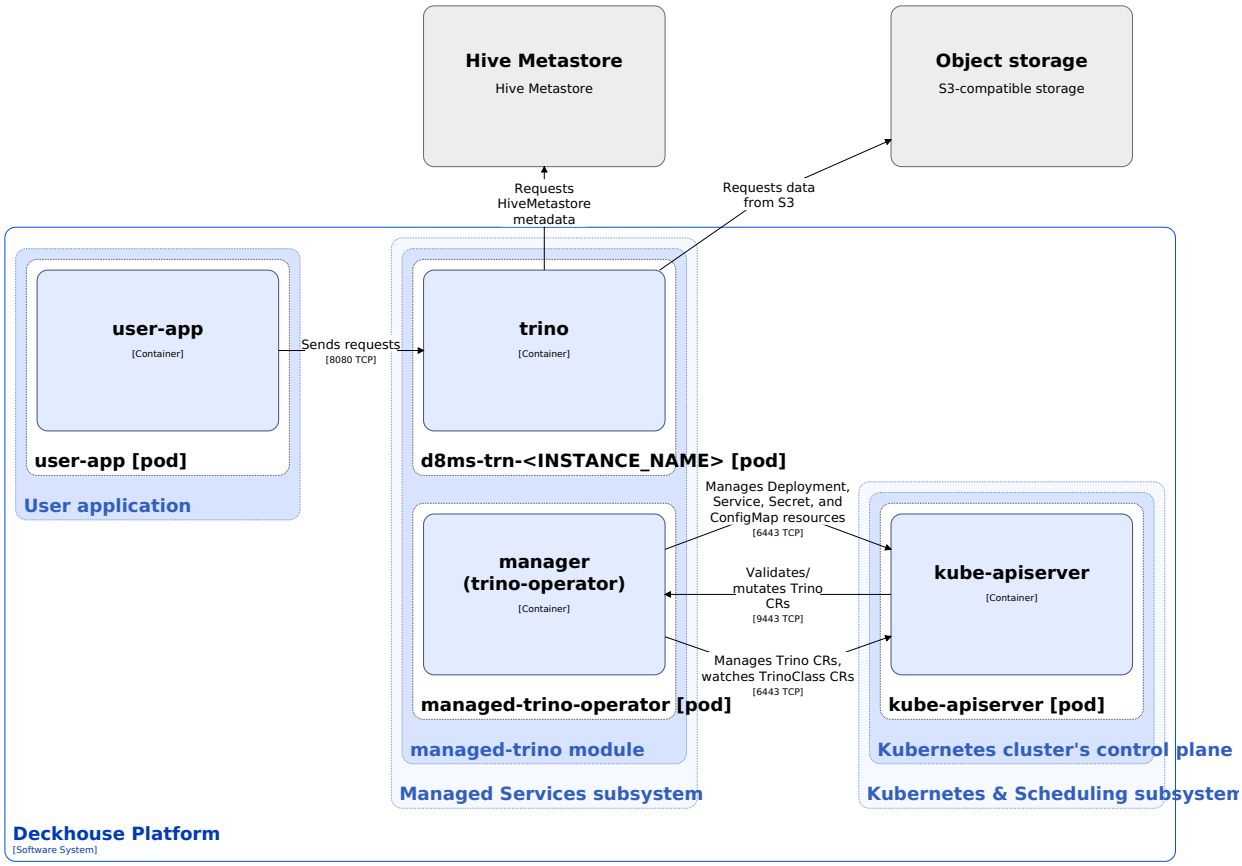
- **Automatic Deployment**: Creates a Trino instance using a simple YAML configuration.
- **Standalone**: Supports deployment of a standalone instance.
- **External Connections**: Supports working with the Hive Metastore metadata store, as well as with S3 storage.
- **Configuration Management**: Separate TrinoClass resource for templating service creation and flexible validation of user parameters.
- **Status**: Displays the current status of the deployed Trino instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-trino` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-trino-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:

    - Reconciles Trino custom resources in all user namespaces. The Trino and TrinoClass custom resources define the settings of the Trino instance.
    - Creates and manages Deployment, Service, Secret, and ConfigMap resources related to the Trino instance.
    - Validates and mutates Trino custom resources using the Validating/Mutating Admission Controllers mechanism.
2. **D8ms-trn-<INSTANCE_NAME>** (Deployment): A component consisting of a single **trino** container that starts and runs the Trino instance. It is created by the managed-trino-operator component for each Trino custom resource.

Module interactions

The module interacts with the following components:

1. **Hive Metastore instance**: Processes data in the metadata store.
2. **Object storage**: Processes data in the object storage.
3. **Kube-apiserver**:
    - Manages Trino custom resources.
    - Watches TrinoClass custom resources.
    - Manages Deployment, Service, Secret, and ConfigMap resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Trino custom resources.
2. **User applications**: Send requests to the Trino instance.

Additional resources

- Module `managed-trino` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managed-valkey module

The `managed-valkey` module manages Valkey instances (a Redis-compatible in-memory data store) in Deckhouse Platform (DP). It provides:

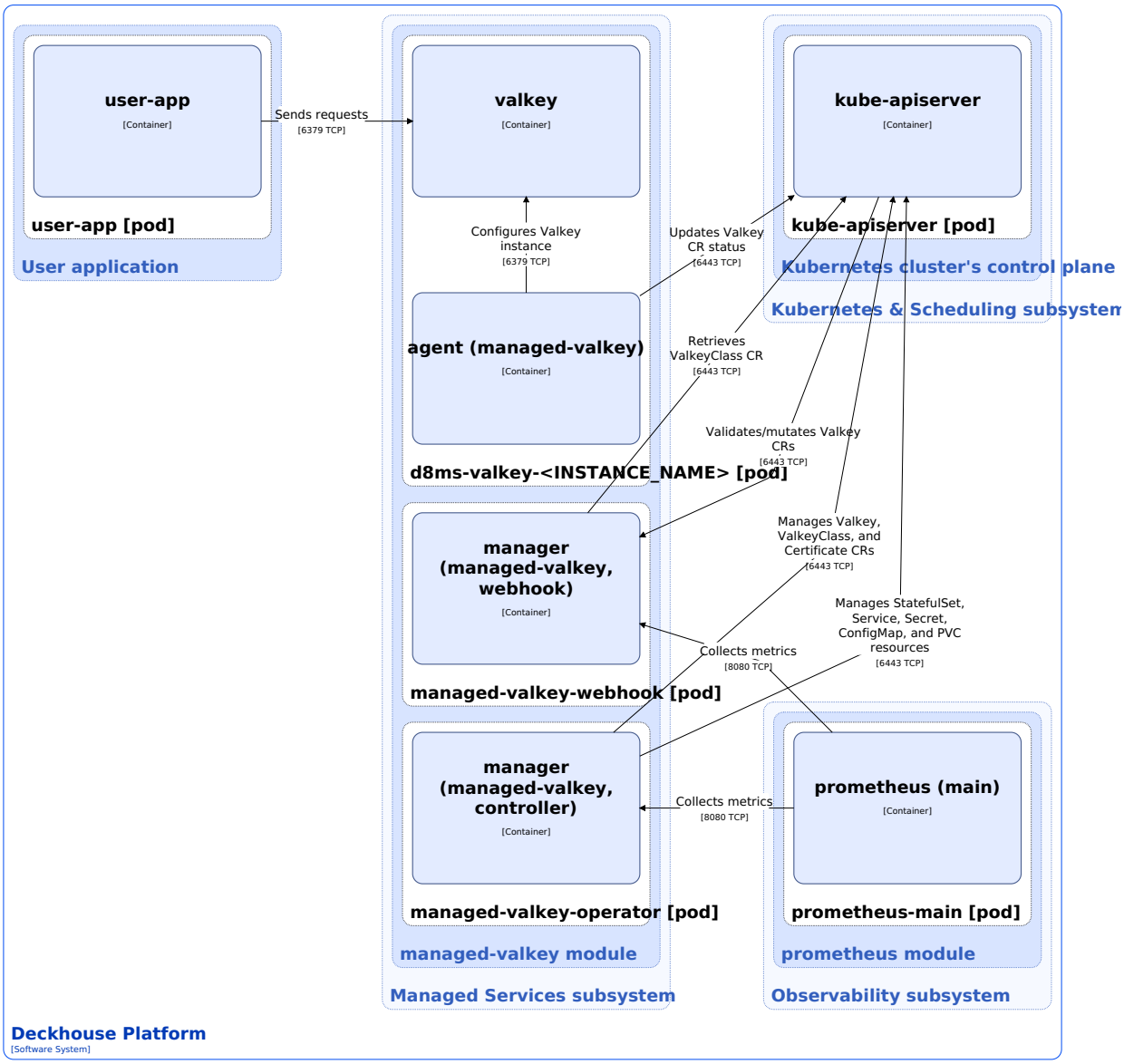
- **Automatic Deployment**: Creates a Valkey instance using a simple YAML configuration.
- **Standalone**: Supports deployment of a standalone instance.
- **Persistent Storage**: Allows configuring different data storage options: `AOF`, `RDB`, `AOF+RDB`.
- **Configuration Management**: Separate ValkeyClass resource for templating service creation and flexible validation of user parameters.
- **Status**: Displays the current status of the deployed Valkey instance.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica of each Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-valkey` module and its interactions with other components of DP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-valkey-operator** (Deployment): Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - It reconciles Valkey custom resources in all user namespaces. The Valkey and ValkeyClass custom resources define the settings of the Valkey instance.
   - It creates and manages StatefulSet, Secret, ConfigMap, and PersistentVolumeClaim resources related to the Valkey instance.
2. **Managed-valkey-webhook** (Deployment): Component consisting of a single container **manager**.

   The managed-valkey-webhook component validates and mutates Valkey custom resources, and mutates ValkeyClass custom resources using the Validating/Mutating Admission Controllers mechanism.
3. **D8ms-valkey-<INSTANCE_NAME>** (StatefulSet): Component that starts and prepares the Valkey instance. It is created by the managed-valkey-operator component.

   It consists of two containers:
   - **valkey**: Main container. It is an open-source project.
   - **agent**: Sidecar container that configures the main container according to the parameters in the Valkey resource.

Module interactions

The module interacts with the **kube-apiserver** component, through which:

- It manages Valkey, ValkeyClass, and Certificate custom resources.
- It manages StatefulSet, Secret, ConfigMap, and PersistentVolumeClaim resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates and mutates Valkey custom resources, mutates ValkeyClass custom resources.
2. **Prometheus-main**: Collects metrics from the managed-valkey-operator and managed-valkey-webhook components.
3. **User applications**: Send requests to the Valkey instance.

Additional resources

- Module `managed-valkey` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtualization subsystem

This subsection describes the architecture of the Virtualization subsystem of Deckhouse Platform (DP).

The Virtualization subsystem is represented by the `virtualization` module, which allows you to declaratively create, run, and manage virtual machines (VM) and their resources.

The `virtualization` module consists of the following components:

- Virtualization API: A controller that implements a user API for creating and managing VM resources.
- A module core: It is based on the KubeVirt project and uses QEMU/KVM + libvirtd to run VMs. KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables a cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane.
- Deckhouse Virtualization Container Registry (DVCR): A container registry for storing and caching VM images.
- Auxiliary components: Components that implement the following auxiliary functions:
  - Security events audit.
  - Forwarding USB devices to VMs.
  - Updating network routes.
  - Deleting resources before deactivating the `virtualization` module.

For more details about the module, refer to the module documentation section.

Additional resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Virtualization API

The Virtualization API component of the `virtualization` module manages custom resources of the following API groups:

1. `virtualization.deckhouse.io`: The main group, it includes the following custom resources:
   - VirtualMachine: A resource that describes the virtual machine (VM) configuration and status.
   - VirtualMachineClass: A resource that describes a set of parameters for VirtualMachine resources, such as CPU and RAM specification, `NodeSelector`, and `Tolerations`.
   - VirtualMachinePool: A resource that declaratively manages a group of identical VMs.
   - VirtualDisk: A resource that describes desired VM disk configuration.
   - VirtualImage: A resource that describes:
     - a VM disk image that can be used as a data source for new VirtualDisk resources.
     - an ISO installation image that can be mounted directly into a VirtualMachine resource.

   The full list of the main API group resources is given in the module documentation.

   Virtualization-controller manages the resources of the main group.

2. `subresources.virtualization.deckhouse.io`: Subresources group. Subresources are additional operations or actions that can be performed on core resources (for example, VirtualMachine) via the Kubernetes API. They provide interfaces for managing specific aspects of resources without affecting the entire object. Instead of the declarative resource familiar to Kubernetes, they are endpoints for imperative operations.

   The group includes the following subresources:

   - `virtualmachines/console`;
   - `virtualmachines/vnc`;
   - `virtualmachines/portforward`;

- `virtualmachines/addvolume`;
- `virtualmachines/removevolume`;
- `virtualmachines/freeze`;
- `virtualmachines/unfreeze`;
- `virtualmachines/addresourceclaim`;
- `virtualmachines/removeresourceclaim`.

Virtualization-api manages subresources.

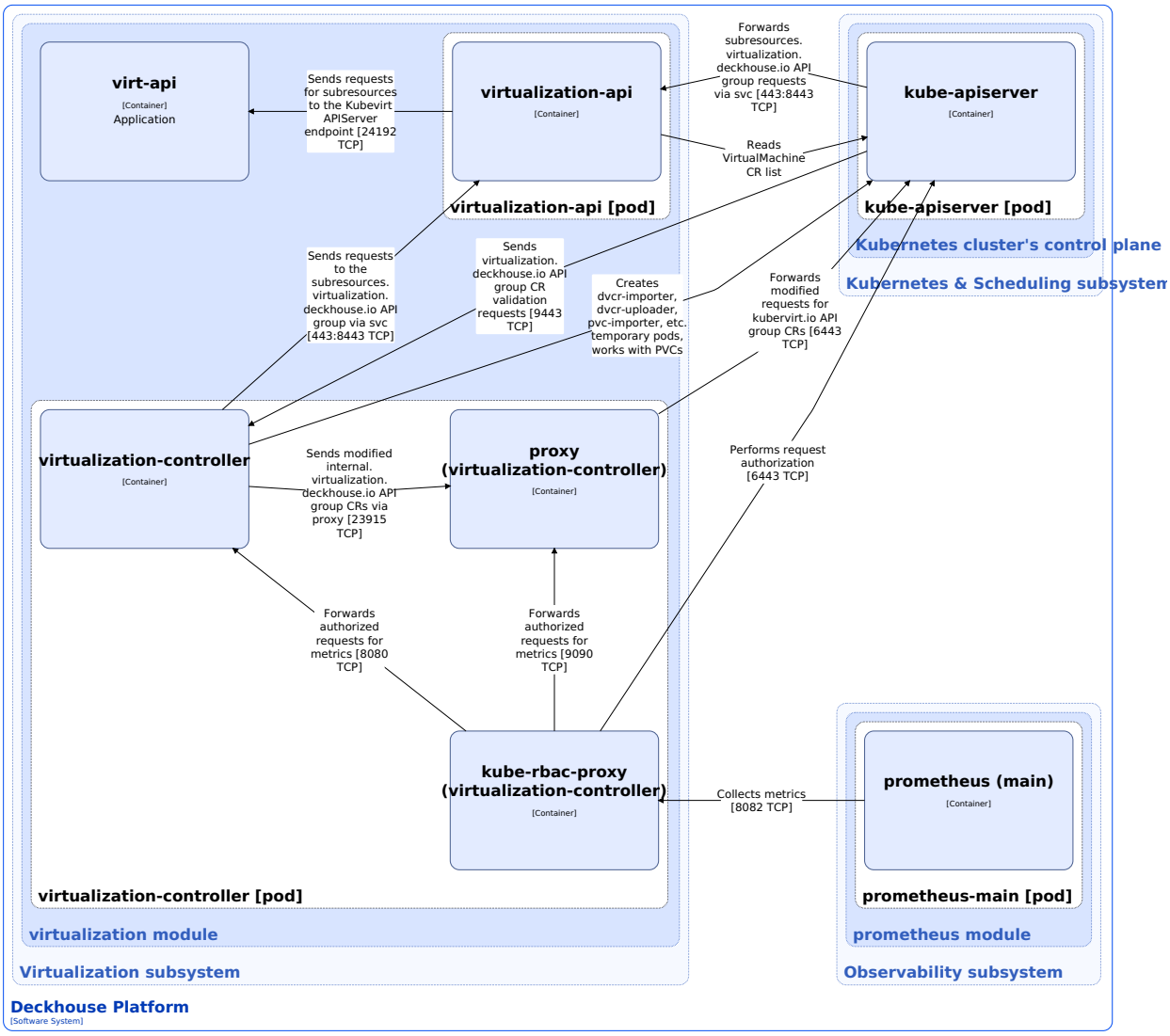
The Virtualization API component of the module uses KubeVirt custom resources as a backend to manage VMs, VM disks, and images.

> **i** KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane.

Virtualization API architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the Virtualization API component of the `virtualization` module and its interactions with other components of Deckhouse Platform (DP) are shown in the following diagram:



Virtualization API components

Virtualization API consists of the following components:

1. **Virtualization-api**: A Kubernetes Extension API Server that serves requests to the `subresources.virtualization.deckhouse.io` API group. Virtualization-api uses `subresources.kubevirt.io` API group subresources as a backend. Virtualization-api accesses the virt-api endpoint directly, which is a Kubernetes Extension API Server that handles requests to similar `subresources.kubevirt.io` API group subresources. It consists of the **virtualization-api** container.

2. **Virtualization-controller**: A controller that performs the following operations:

   - Manages custom resources of the main `virtualization.deckhouse.io` API group. Virtualization-controller will not change the main portion of these custom resources: `labels`, `annotations`, and `spec`. Virtualization-controller can make the following changes to custom resources:

     - Adding and removing finalizers in the `metadata.finalizers` attribute.
     - Adding and removing entries in the `metadata.ownerReferences` attribute.
     - Modifying the `status` subresource.

     Virtualization-controller uses `kubevirt.io` API group custom resources as a backend.

   - Validation of `virtualization.deckhouse.io` API group resources using the Validating Admission Controllers mechanism.
   - Launching `dvcr-importer`, `dvcr-uploader` and `pvc-importer` pods to import and upload VM disks and images. For more details, see Importing and uploading VM images and disks.
   - Performing operations on VMs by making requests to some subresources of the `subresources.virtualization.deckhouse.io` API group, for example `virtualmachines/freeze` and `virtualmachines/unfreeze`.

   The component consists of the following containers:

   - **virtualization-controller**: Main container that implements the controller and webhook server.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it, namely renaming the metadata of custom resources. This is necessary because KubeVirt components use API groups like `*.kubevirt.io`, and other components of the `virtualization` module use similar resources, but with API groups like `*.virtualization.deckhouse.io`. Kube-api-rewriter is a gateway that proxies requests between controllers that manage resources from different API groups and is an open-source project.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the controller and the proxy sidecar container. It is an open-source project.

Virtualization API interactions

Virtualization-api interacts with the following components:

1. **Kube-apiserver**: Lists VirtualMachine custom resources, which are needed to process requests to subresources.
2. **Virt-api**: Sends requests to the KubeVirt subresources. Requests pass through a similar proxy sidecar container that renames metadata from the `subresources.virtualization.deckhouse.io` API group to the `subresources.kubevirt.io` API group and proxies them to the virt-api endpoint (Kubernetes Extension API Server KubeVirt).

Virtualization-controller interacts with the following components:

1. **Kube-apiserver**:

   - Sends modified virtualization module custom resources via a proxy sidecar container that renames `internal.virtualization.deckhouse.io` API group resource metadata to `kubevirt.io` API group resource metadata.
   - Authorizes requests for metrics.

The following external components interact with the Virtualization API component:

1. **Kube-apiserver**:

   - Forwards requests to `subresources.virtualization.deckhouse.io` API group subresources.
   - Sends `virtualization.deckhouse.io` API group resource validation requests.

2. **Prometheus-main**: Collects component metrics.

Additional resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The virtualization core

The `Virtualization` module core is directly responsible for working with virtual machines (VMs). The core is based on the KubeVirt project. KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables a cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane. A fork of KubeVirt from the company “Flant” is used in the `virtualization` module.

To manage VMs, the module core uses custom resources of the following API groups:

1. `internal.virtualization.deckhouse.io`: The main group, an analog of the `kubevirt.io` API group of the original KubeVirt. It includes the following custom resources:

   - InternalVirtualizationVirtualMachine: A resource that describes the VM configuration and status.
   - InternalVirtualizationVirtualMachineInstance: A resource that describes a running VM. After VM shutdown, the InternalVirtualizationVirtualMachineInstance resource is deleted, but the InternalVirtualizationVirtualMachine resource that manages the InternalVirtualizationVirtualMachineInstance lifecycle remains.

   The resources of the main group are managed by the virt-controller component. The InternalVirtualizationVirtualMachine resources of the KubeVirt `internal.virtualization.deckhouse.io` main API group are used as the backend for the VirtualMachine resources of `virtualization.deckhouse.io` API group managed by the virtualization-controller.

   > **i** For simplification, the reduced VirtualMachine and VirtualMachineInstance names will be used for the InternalVirtualizationVirtualMachine and InternalVirtualizationVirtualMachineInstance resources (of the original KubeVirt `kubevirt.io` API group) respectively.

2. `subresources.kubevirt.io`: Subresources group. Subresources are additional operations or actions that can be performed on core resources (for example, VirtualMachineInstance) via the Kubernetes API. They provide interfaces for managing specific aspects of resources without affecting the entire object. Instead of the declarative resource familiar to Kubernetes, they are an endpoint for imperative operations. The following KubeVirt subresources are used in the `virtualization` module:

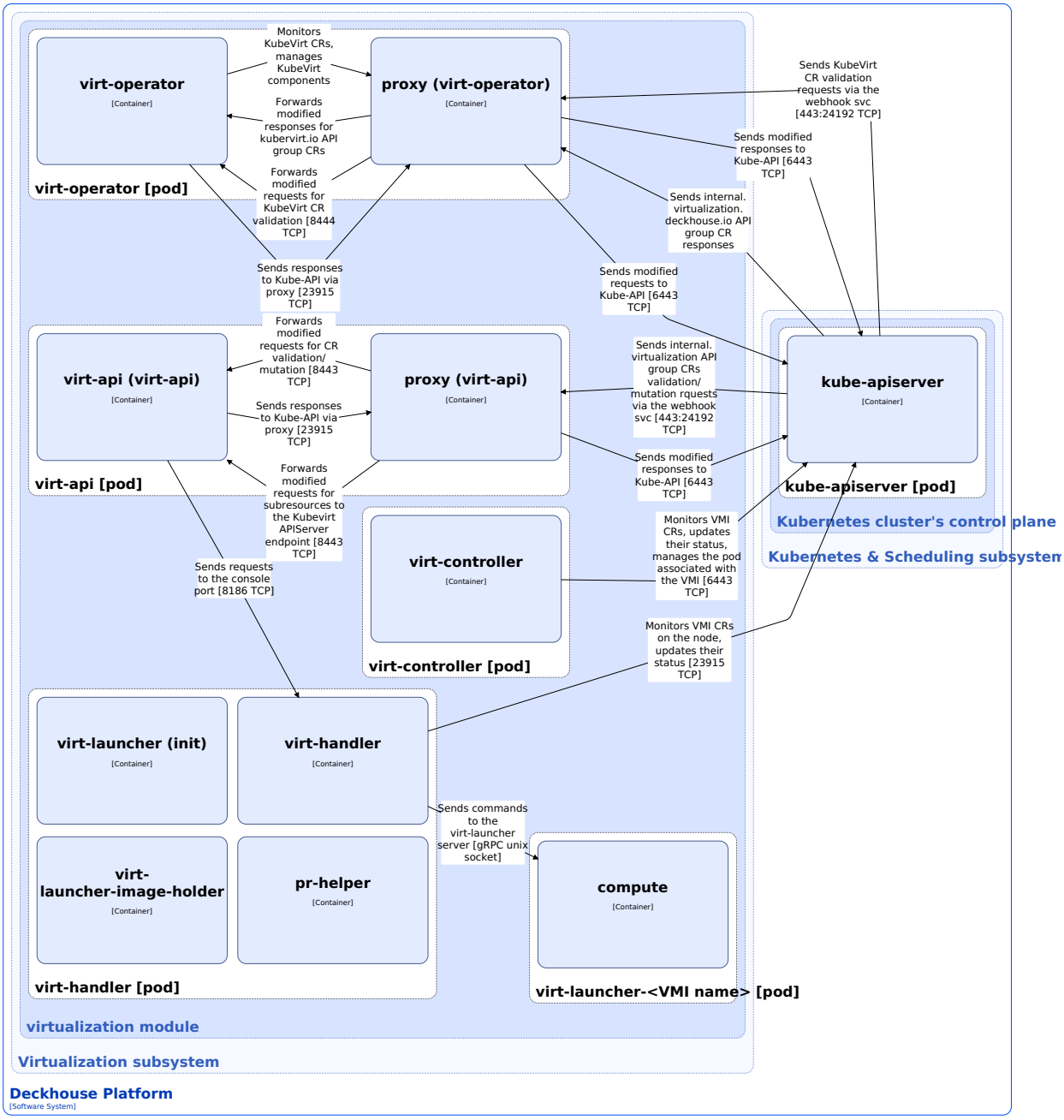
   - `virtualmachines/{name}/addvolume`;
   - `virtualmachines/{name}/removevolume`;
   - `virtualmachines/{name}/addresourceclaim`;
   - `virtualmachines/{name}/removeresourceclaim`;
   - `virtualmachineinstances/{name}/console`;
   - `virtualmachineinstances/{name}/vnc`;
   - `virtualmachineinstances/{name}/portforward`;
   - `virtualmachineinstances/{name}/freeze`;
   - `virtualmachineinstances/{name}/unfreeze`.

   The subresources are managed by the virt-api component. The KubeVirt subresources listed above are used as a backend for similar resources of `subresources.virtualization.deckhouse.io` API group managed by the virtualization-api component.

Module core architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `virtualization` module core and its interactions with other components of DP are shown in the following diagrams:



Module core components

The module core consists of the following components:

1. **Virt-api**: A Kubernetes Extension API Server that serves requests to the `subresources.kubevirt.io` API groups. The virt-api component performs validation and mutation of custom resources of the `internal.virtualization.deckhouse.io` API groups using the Validating/Mutating Admission Controllers mechanism. Requests pass through the **proxy** sidecar container that renames metadata from the `internal.virtualization.deckhouse.io` API group to the `kubevirt.io` API group and proxies them to the virt-api component endpoint.

   It consists of the following containers:

   - **virt-api**: Main container that implements controller and webhook server.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it, namely renaming the metadata of custom resources. This is necessary because KubeVirt components use API groups like `*.kubevirt.io`, and other components of the `virtualization` module use similar resources, but with API groups like `*.virtualization.deckhouse.io`. Kube-api-rewriter is a gateway that proxies requests between controllers that manage resources from different API groups.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the proxy container. It is an open-source project.

2. **Virt-controller**: A controller that manages the `internal.virtualization.deckhouse.io` main API group custom resources and responsible for the virtualization functionality at the cluster wide level. For each VirtualMachineInstance resource it creates a separate pod in which the VM is started. Virt-controller monitors VirtualMachineInstance resources, updates their status, and manages the associated pods.

   It consists of the following containers:

- **virt-controller**: Main container.
- **proxy** (aka **kube-api-rewriter**): A sidecar container that performs modification of API requests passing through it (described above).
- **kube-rbac-proxy**: A sidecar container providing authorized access to the metrics of the virt-controller and proxy containers (described above).

3. **Virt-handler** (DaemonSet): A separate controller that runs on all nodes of the cluster. Virt-handler performs the following functions:

   - It extends the kubelet functionality, configuring the pod environment to run a VM inside it. At the moment virt-handler is responsible for creating network interfaces, also it is used to forward `dev/kvm` and other devices from the node into the pod. Virt-handler uses kubelet device plugins to forward devices.

   - Like virt-controller, virt-handler watches VirtualMachineInstance resources corresponding to VMs running on the node. When changes are detected, virt-handler sends a command to the virt-launcher process running in the compute container of the VM pod. Virt-launcher changes the VM state in accordance with the command received. Virt-handler also monitors VM events that virt-launcher returns and synchronizes the corresponding VirtualMachineInstance resource status.

   - It accepts commands from the virt-api component via the `console-port` that correspond to requests for subresources and forwards them to the virt-launcher for execution. By the subresources functionality ports are forwarded to the VM, as well as to the regular and VNC consoles.

     Virt-handler interacts with virt-launcher via the gRPC protocol and Unix socket.

   It consists of the following containers:

   - **virt-launcher**: Init container that runs `node-labeller.sh` script via virt-launcher binary. This script prepares CPU parameters, their functions, and machine types which virt-handler will use to set the appropriate labels on Node resources. These labels, in turn, will be used to schedule VMs to the nodes that support the appropriate parameters.
   - **virt-handler**: Main container.
   - **virt-launcher-image-holder**: A service sidecar container for virt-launcher image pre-loading. The container is on pause and performs only the function of storing the image.
   - **pr-helper**: QEMU persistent reservation helper is a service sidecar container that creates a listener socket that accepts incoming connections to communicate with QEMU. This is necessary because the operating system restricts sending SCSI commands with permanent redundancy to unprivileged programs, which prevents multiple VMs from sharing block SCSI devices, for example in the case of clusterization. QEMU is a free and open-source program for emulating the hardware of various platforms, which is used to run VMs in a pod.

4. **Virt-operator**: A Kubernetes operator that manages the KubeVirt components lifecycle using InternalVirtualizationKubeVirt custom resource. Virt-operator installs the virt-api, virt-controller and virt-handler in the cluster and also configures them.

   It consists of the following containers:

   - **virt-operator**: Main container.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it (described above).

5. **Virt-launcher-<VMI name>**: The pod where the VM is running (more exactly VirtualMachineInstance).

   It consists of one container:

   - **compute**: A container where the virt-launcher is run. Virt-launcher implements cmd-server (a gRPC server for the remote command execution).

     Depending on the command received from virt-handler, virt-launcher generates an XML specification of the VM being launched or updated and sends it to libvirtd. Libvirtd is a daemon of the server part of the libvirt virtualization management system. It runs on host servers and performs management tasks for virtual guest systems.

     Libvirtd, in turn, starts the VM and manages its lifecycle. The VM is started using QEMU and KVM. QEMU is an open-source emulator that supports hardware virtualization and works in conjunction with the KVM hypervisor.

     In fact libvirtd starts the QEMU process, which is the VM (more exactly VirtualMachineInstance).

     Virt-handler also constantly monitors the status of the running VM, returned by libvirtd via virt-launcher, and updates the VirtualMachineInstance status.

Module core interactions

The module core interacts with the following components:

1. **Kube-apiserver**:

   - Watches for KubeVirt custom resources and manages KubeVirt components.
   - Watches for VirtualMachineInstance custom resources, updates their status, and manages the associated pods.
   - Authorizes requests for metrics.

The following external components interact with the module core:

1. **Kube-apiserver**:

   - Sends InternalVirtualizationKubeVirt custom resource validating requests.
   - Sends `internal.virtualization.deckhouse.io` API group resource validating and mutating requests.

2. **Prometheus-main**: Collects module core components metrics.

Additional resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse Virtualization Container Registry (DVCR)

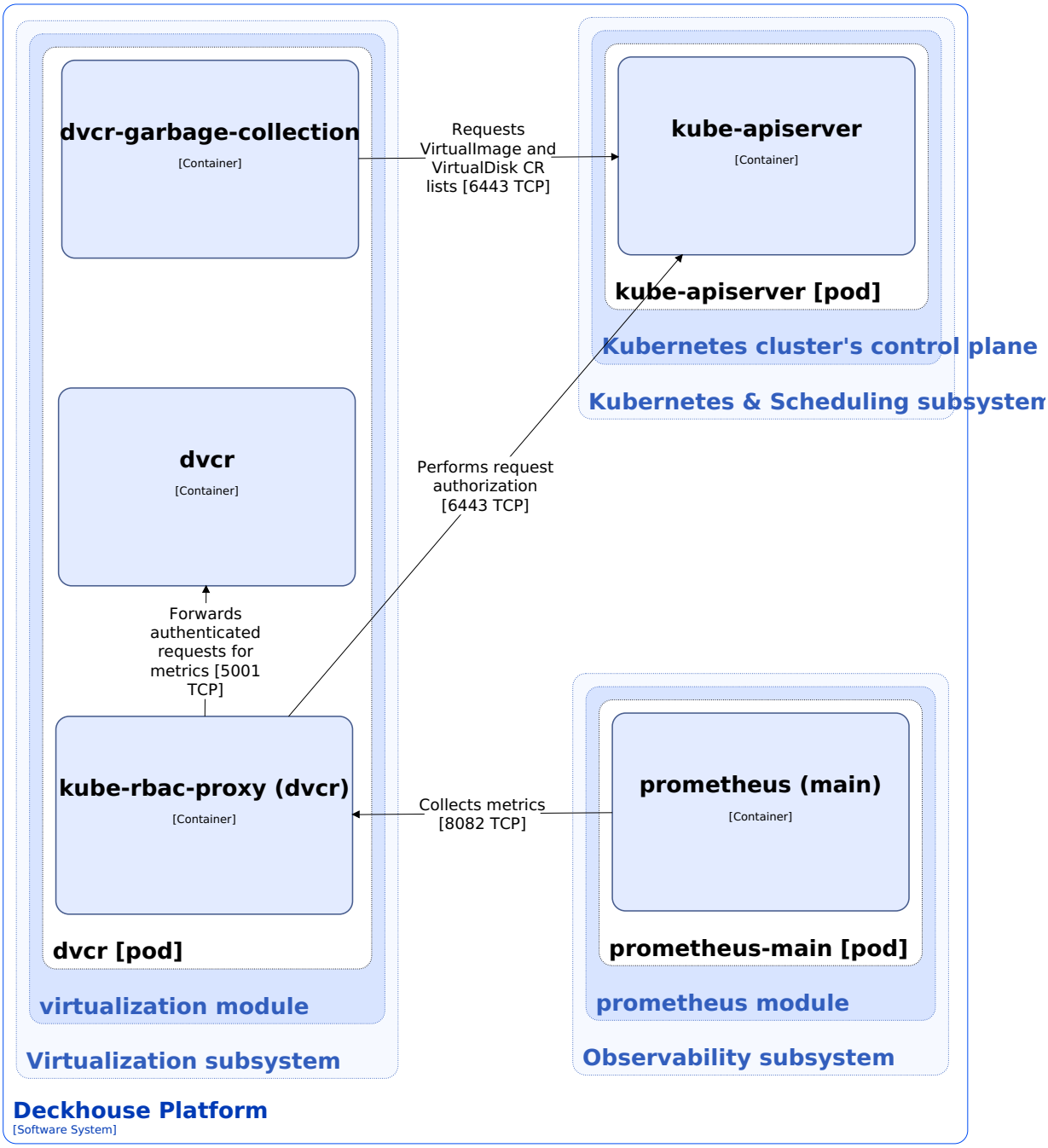
The Deckhouse Virtualization Container Registry (DVCR) component of the `virtualization` module is a specialized container registry for storing and caching virtual machine (VM) images. Virtualization-controller of the Virtualization-API component of the `virtualization` module allows you to import images stored in DVCR into PVC volumes used as VM disks managed by KubeVirt. For more details, see Importing and uploading VM images and disks.

> KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane.

DVCR architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the DVCR component of `virtualization` module and its interactions with other components of DP are shown in the following diagrams:



DVCR components

DVCR consists of the following components:

1. **Dvcr**: A container registry based on Distribution. Distribution is an open-source project that provides a framework for storing and distributing container images and other content using the OCI Distribution Specification. Dvcr is used for storing and caching VM images.

   It consists of the following containers:

   - **dvcr**: Main container.
   - **dvcr-garbage-collection**: Sidecar container that periodically deletes images which do not have the appropriate resources in the cluster.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the dvcr container. It is an open-source project.

DVCR interactions

DVCR interacts with the following components:

1. **Kube-apiserver**: Sends `get`/`list`/`watch`-requests for VirtualImages, ClusterVirtualImages, and VirtualDisks to clean up unused images and for coordination.

The following external components interact with the DVCR component:

1. **Prometheus-main**: Collects metrics of the dvcr component.

Additional resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Importing and uploading VM images and disks

Virtualization-controller of the Virtualization-API component of the `virtualization` module allows you to import VM images and disks from different sources into PVC volumes used as VM disks managed by KubeVirt.

> KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane.

Depending on the scenarios for importing or uploading VM images and disks, virtualization-controller launches the following *temporary* pods:

1. **D8v-<vi|cvi>-importer**: Pod consisting of a single **d8v-dvcr-importer** container, that is launched to implement the following scenarios for importing VM images and disks to DVCR storage:
   - Import of a VM image from external sources (HTTP source available via URL or container registry).
   - Import of a VM image, disk or snapshot from VirtualImage, ClusterVirtualImage, VirtualDisk or VirtualDiskSnapshot resources.
2. **D8v-<vi|cvi>-uploader**: Pod consisting of a single **d8v-dvcr-uploader** container, that is launched to upload VM images and disks to DVCR storage *by the user*.
3. **D8v-<vd|vi>-pvc-importer**: Pod consisting of a single **d8v-pvc-importer** container, that is launched to import VM image from container registry (DVCR) into PVC volume. Only KubeVirt data import is supported, meaning the imported image must be a KubeVirt VM disk. If necessary, d8v-pvc-importer will automatically decompress and convert the file from the supported format to `raw` or `qcow2` format (depending on the volume mode). It will also resize the disk to use all available space. The data during the import process is saved to a temporary prime PVC. After successfully copying the data to the prime PVC, the virtualization-controller binds the target PVC to the PV of the prime PVC, and then the prime PVC as well as original PV of target PVC are deleted.
4. **D8v-pvc-pvc-source-importer** and **d8v-pvc-pvc-target-importer**: Pod consisting of a single **d8v-pvc-source-importer** container and pod consisting of a single **d8v-pvc-target-importer** container, respectively, that are launched to import a VM image from one PVC volume to another PVC volume (cloning over the network). The PVC source is mounted as a network block device using the Network Block Device (NBD) protocol in the d8v-pvc-pvc-source-importer pod, which is accessed by an application running in the d8v-pvc-pvc-target-importer pod container. The data during the import process is saved in a temporary primary (prime) PVC. After successfully copying the data to the primary PVC, the virtualization-controller binds the target PVC (PVC receiver) to the PV of the primary PVC, and then the prime PVC as well as original PV of target PVC are deleted.

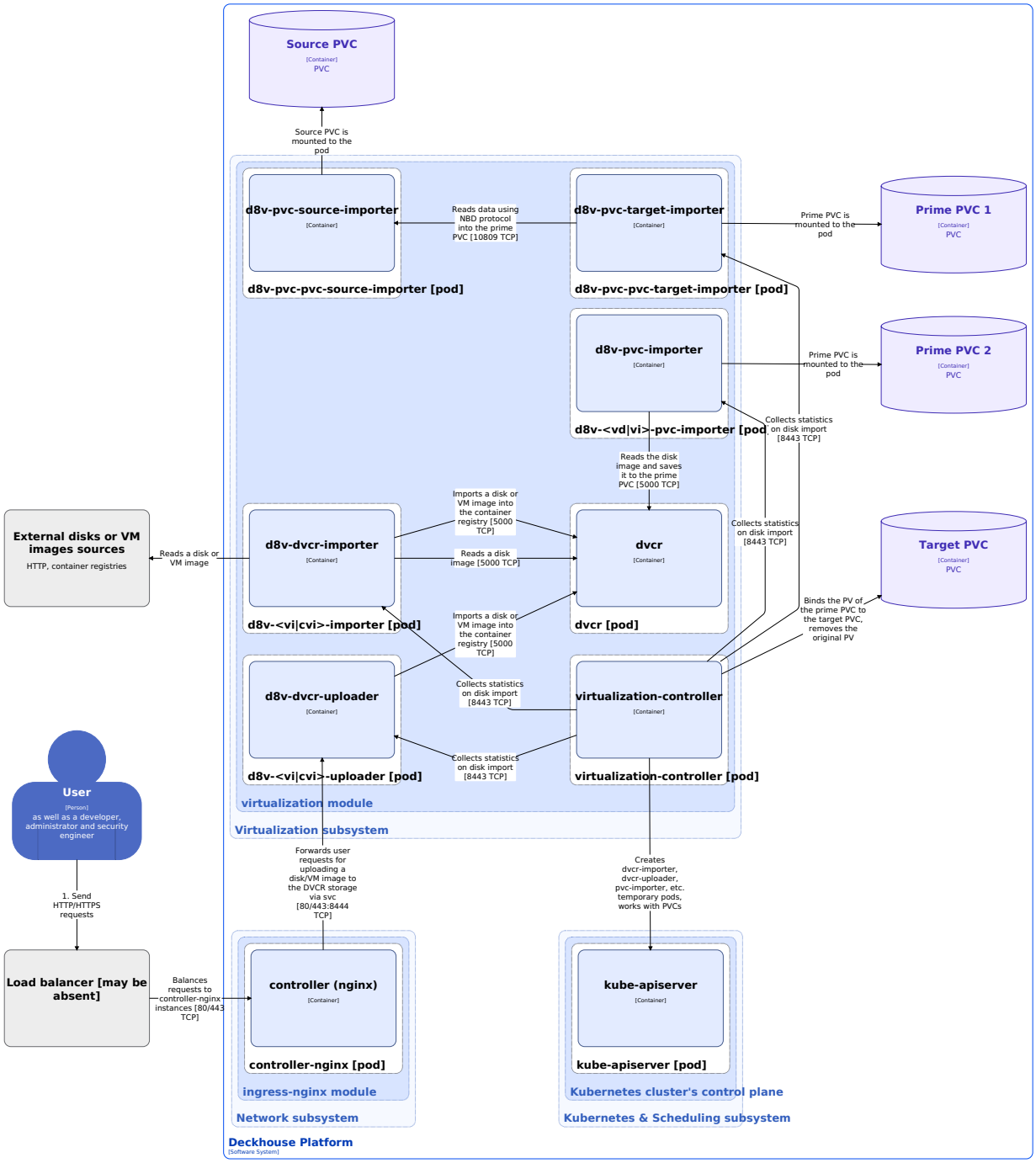
During the import of a VM image or disk, virtualization-controller collects import statistics directly from the running temporary pod and, based on these statistics, calculates the download progress in percentage terms, as well as the average and current download speeds, to be displayed in the status of the corresponding custom resource (VirtualImage, ClusterVirtualImage, or VirtualDisk).

Virtualization-controller also supports the following scenarios for importing VM images and disks that do not require launching temporary pods:

- Importing a VM image from container registry into PVC volume.
- Importing a VM image from volume snapshot (VolumeSnapshot resource) into PVC volume.
- Importing a VM image from one PVC volume to another PVC volume (cloning by CSI driver).
- Creating an empty PVC volume.

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

Interactions of `virtualization` module components while importing and uploading VM images and disks are shown in the following diagram:



During the processes of importing and uploading VM images and disks `virtualization` module components interact with the following external components:

1. **External sources of VM disks and images**: Read disks or VM images when implementing some scenarios for importing to DVCR storage.
2. **Ingress-controller**: Forwards user requests to upload VM image or disk to DVCR storage via HTTP endpoint of dvcr-uploader service.

Additional resources

- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Auxiliary components

The `virtualization` module uses components that implement the following auxiliary functions:

- Security events audit.
- Forwarding USB devices to virtual machines (VMs).

- Updating network routes.
- Deleting resources before deactivating the `virtualization` module.

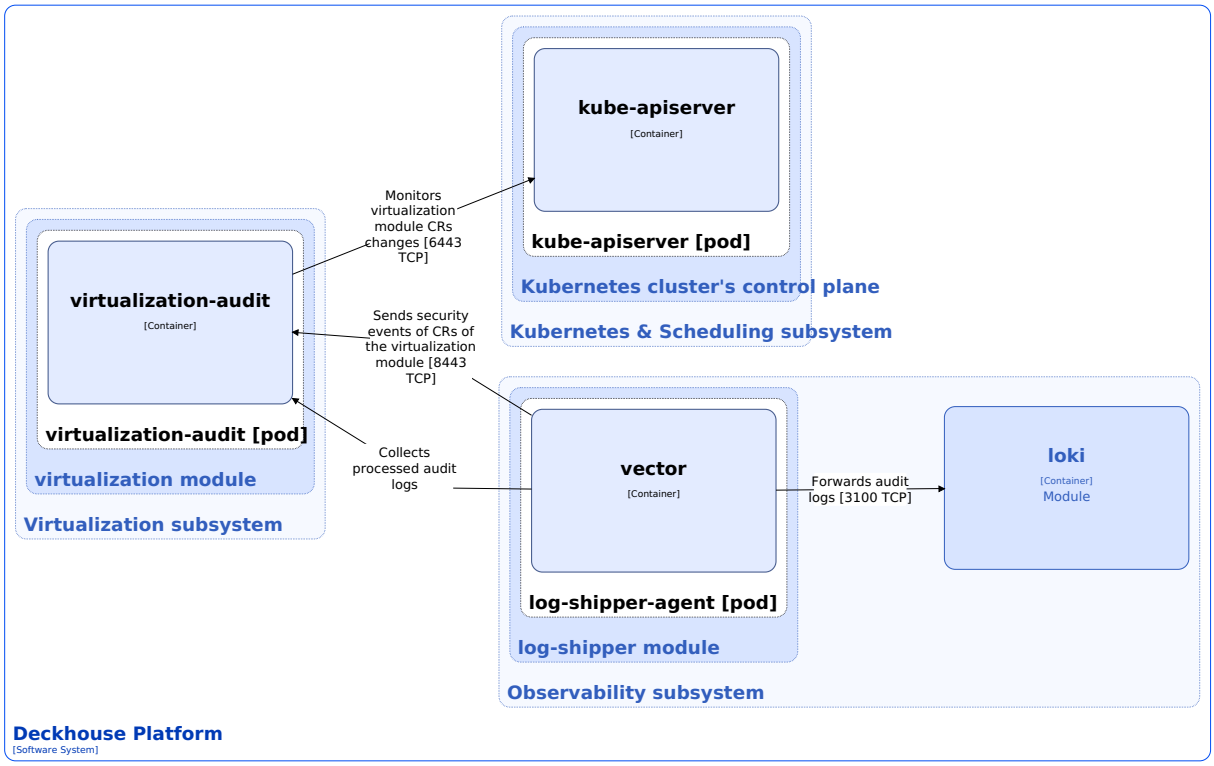
Security events audit

For instructions on `virtualization` module security events audit activation, refer to the module documentation.

Architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `virtualization` module auxiliary components for security events audit and their interactions with other components of DP is shown in the following diagram:



Components

Virtualization-audit is a component consisting of a single container that receives the `virtualization` module security events stream. Sending events is implemented by the `log-shipper` module. The Vector logging agent, according to the settings in the ClusterLoggingConfig custom resources, selects events related to the `virtualization` module custom resources from the cluster's audit log and sends them to the virtualization-audit service endpoint. Virtualization-audit processes the received audit events, enriches them with data from the Kubernetes API, and saves the processed events to its own log.

You can forward security events to the cluster logging system (for example, Loki). In this case, ClusterLoggingConfig resources and the vector agent of the `log-shipper` module are used in a similar way.

Interactions

Virtualization-audit interacts with the following components:

1. **Kube-apiserver**: Watches for `virtualization` module custom resources.

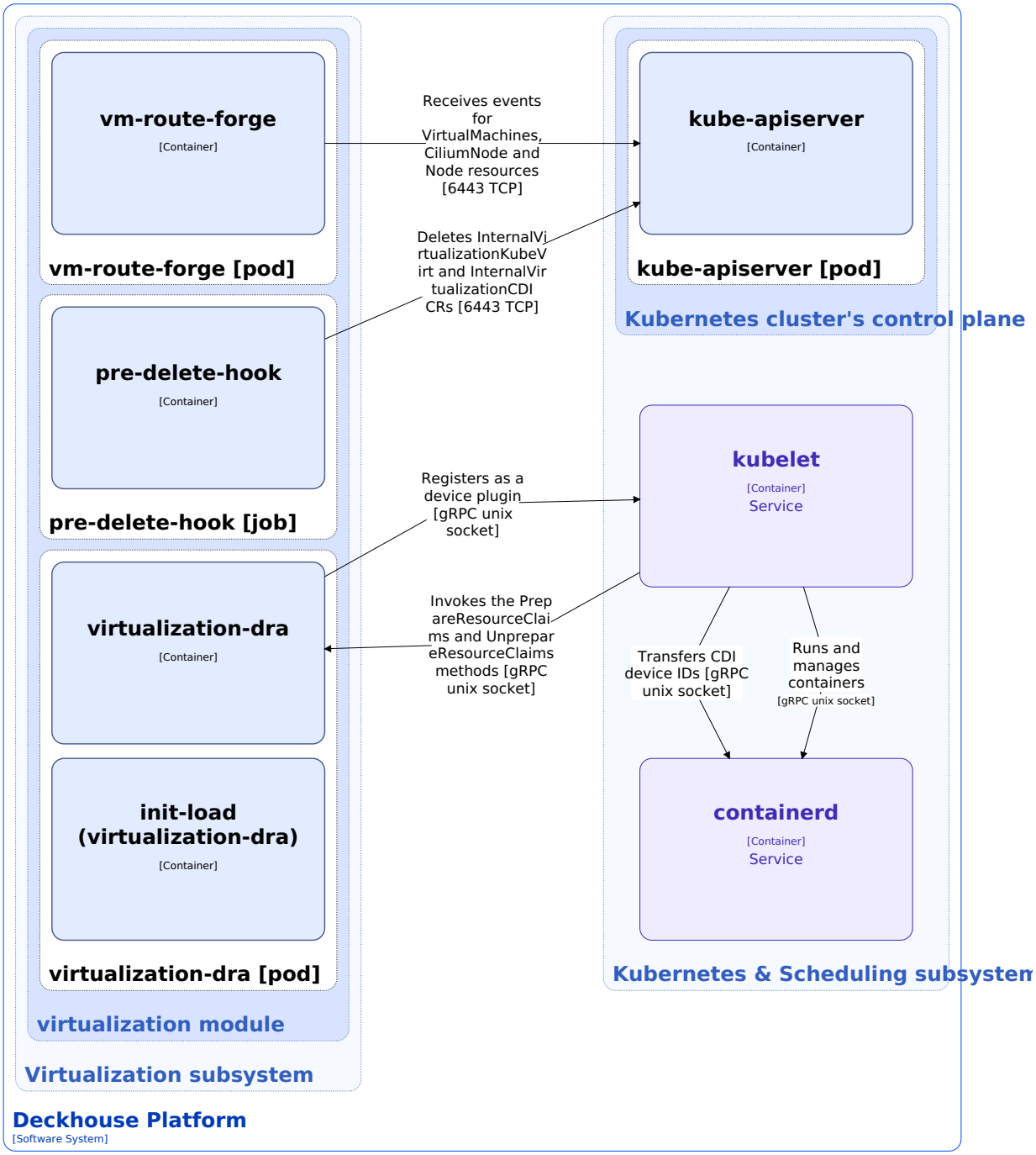
The following external components interact with virtualization-audit:

1. **Log-shipper-agent**:
   - Sends `virtualization` module security events.
   - Collects processed audit logs.

Virtualization-dra and other components

Architecture

The Level 2 C4 architecture of other `virtualization` module auxiliary components and their interactions with other components of DP is shown in the following diagram:



Components

1. **Virtualization-dra** (DaemonSet): The DRA driver that forwards USB devices to VMs. The DRA (Dynamic Resource Allocation) technology is used to forward USB devices. DRA is a Kubernetes API, scheduler, and kubelet engine for describing, scheduling, and provisioning dynamically allocated resources through external drivers. The DRA driver performs the following operations:
   - Automatically detects USB devices on cluster nodes and publishes them as a ResourceSlice resource. virtualization-controller synchronizes this data into NodeUSBDevice custom resources, which are then used to configure USB device forwarding. For more information about configuring USB device forwarding, refer to the module documentation.
   - Registers in kubelet as the DRA kubelet plugin. The DRA kubelet plugin prepares and releases allocated resources for pods through the PrepareResourceClaims and UnprepareResourceClaims operations. The PrepareResourceClaims method returns CDI (Container Device Interface) device IDs that kubelet passes to containerd. Data on available USB devices is published via ResourceSlice, and device selection is performed by the Kubernetes DRA mechanism based on ResourceClaim/ResourceClaimTemplate and DeviceClass.

     The DRA driver interacts with kubelet via the gRPC protocol and Unix sockets.
   - Implements a USBIP server, so that the USB device is automatically forwarded over the network to the node where the VM is running. There is no need to manually place the VM on the same node where the device is located.

It consists of the following containers:

- **init-load**: Init container that loads the Linux kernel modules necessary for the DRA driver operation.
- **virtualization-dra**: Main container.

2. **Vm-route-forge**: A controller that monitors VirtualMachine custom resources of the `virtualization.deckhouse.io` API group and updates network routes on the node via Linux netlink/eBPF in routing tables used by CNI Cilium for routing traffic between VMs.
3. **Pre-delete-hook** (Job): A Job started by the Deckhouse controller before the `virtualization` module is deleted. It removes the InternalVirtualizationKubeVirt custom resource named `config`.

Interactions

Virtualization-dra interacts with the following components:

1. **Kubelet**: Registers in kubelet as the DRA kubelet plugin.

Vm-route-forge interacts with the following components:

1. **Kube-apiserver**: Receives events on VirtualMachine, CiliumNode and Node resources.
2. **Host network/Linux kernel**: Updates routes and routing rules on the node.
3. **Cilium data plane**: Uses data from the CiliumNode and Cilium routing tables to route VM traffic.

Pre-delete-hook interacts with the following components:

1. **Kube-apiserver**: Deletes InternalVirtualizationKubeVirt resource named `config`.

The following external components interact with Virtualization-dra:

1. **Kubelet**: Calls PrepareResourceClaims and UnprepareResourceClaims gRPC methods to prepare and release resources associated with USB devices.

Additional resources

- Module `cni-cilium` documentation
- Module `log-shipper` documentation
- Module `virtualization` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Global configuration

Global configuration settings allow you to customize parameters that are used by default by all modules and components of the Deckhouse Platform. Some modules may override some of these parameters (this can be found in the settings section of the respective module's documentation).

The global configuration settings are stored in the ModuleConfig `global`.

> The publicDomainTemplate parameter specifies a DNS name template used by some Deckhouse modules to create Ingress resources. If this parameter is not specified, Ingress resources will not be created.
>
> If you don't have access to wildcard DNS records, you can use sslip.io or similar services for testing purposes.
>
> The domain specified in the template must not match or be a subdomain of the domain specified in the `network.clusterDomain` parameter of the `control-plane-manager` module. We do not recommend changing the cluster domain unless absolutely necessary.
>
> For the template to function correctly, you must first configure DNS services both in the networks where the cluster nodes will be located and in the networks from which clients will access the platform's service web interfaces.
>
> If the template matches the domain of the node network, use only A records to assign addresses of the nodes' Frontend interfaces to the platform's service web interfaces.
> For example, if the nodes are registered under the `company.my` zone and the template is `%s.company.my`.

Example of the `ModuleConfig/global`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings: # <-- Module parameters from the "Parameters" section below.
    defaultClusterStorageClass: 'default-fast'
    modules:
      publicDomainTemplate: '%s.kube.company.my'
      placement:
        customTolerationKeys:
        - dedicated.example.com
      storageClass: 'default-fast'
```

Conversions

Global configuration settings are configured using the `global` ModuleConfig, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Move `storageClass` to `modules.storageClass`. Remove `modules.resourcesRequests.everyNode`, `modules.resourcesRequests.masterNode` and `modules.proxy` parameters.

- Updates from version **2** to **3**:

  Move `modules.ingressClass` to `modules.ingress.ingressClass`, and `modules.gatewayAPIGateway` to `modules.gatewayAPI.gateway`.

Parameters

Schema version: 3

- **settings**
  object
  - **settings.applications**
    object

    Common parameters of the Deckhouse ecosystem applications.

    - **settings.applications.https**
      object

      HTTPS access implementation method used by the Deckhouse ecosystem applications.

      Examples:

      ```
      https:
        certManager:
          clusterIssuerName: letsencrypt
        mode: CertManager
      ```

      ```
      https:
        mode: Disabled
      ```

      ```
      https:
        mode: OnlyInURI
      ```

      ```
      https:
        mode: CustomCertificate
        customCertificate:
          secretName: plainstring
      ```

      - **settings.applications.https.certManager**
        object
        - **settings.applications.https.certManager.clusterIssuerName**
          string

          ClusterIssuer name used by applications to obtain TLS certificates.

          The `cert-manager` module offers the following ClusterIssuer options:

          - `letsencrypt`
          - `letsencrypt-staging`
          - `selfsigned`

    - `clouddns`
    - `cloudflare`
    - `digitalocean`
    - `route53`

  You can use your own ClusterIssuer if necessary.

  Default: `letsencrypt`

- **settings.applications.https.customCertificate**
  object
  - **settings.applications.https.customCertificate.secretName**
    string

    Name of the Secret in the `d8-system` namespace containing a TLS certificate for applications.

    This Secret must be in the `kubernetes.io/tls` format.

    Default: `false`

- **settings.applications.https.mode**
  string

  HTTPS usage mode:

  - `CertManager`: Applications use HTTPS and obtain certificates automatically using the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate`: Applications use HTTPS with the certificate from the `d8-system` namespace.
  - `Disabled`: Applications use HTTP only.
  - `OnlyInURI`: Applications use HTTP in the expectation that there is an external upstream HTTPS load balancer, which terminates HTTPS. This load balancer should be capable of redirecting requests from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.applications.ingressClass**
  string

  Default IngressClass name used by the Deckhouse ecosystem applications when creating Ingress resources.

  Some applications support their own `ingressClass` parameter, which lets you use a different IngressClass instead of the default one.

  Default: `nginx`

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: nginx
  ```

- **settings.applications.publicDomainTemplate**
  string

  DNS name template with two `%s` keys used as the dynamic parts of the string.

  Whenever a DNS is generated:

  - First key is replaced with the application instance name.
  - Second key is replaced with the application’s namespace.

  The Deckhouse ecosystem applications use this template when creating Ingress resources.

  For example, given the template `%s.%s.kube.company.my`, the application instance named `grafana` running in the `monitoring` namespace will get an Ingress resource at the address `grafana.monitoring.kube.company.my`.

  Usage notes:

  - If this parameter is not specified, Ingress resources will not be created.
  - The domain specified in the template must not match or be a subdomain of the domain specified in the `network.clusterDomain` parameter of the `control-plane-manager` module. Changing the cluster domain is not recommended unless necessary.
  - Before using the template, configure DNS services both in the networks where the cluster nodes will be located and in the networks from which clients will access the service web interfaces.
  - If the template domain matches the domain of the node network, use only A records to assign addresses of the frontend nodes to the platform service web interfaces. For example, if the nodes are registered under the `company.my` zone, the template must be `%s.%s.company.my`.

  Pattern: `^(%s([-a-z0-9]*[a-z0-9])?|[a-z0-9]([-a-z0-9]*)?%s([-a-z0-9]*)?[a-z0-9]|[a-z0-9]([-a-z0-9]*)?%s)\.(%s([-a-z0-9]*[a-z0-9])?|[a-z0-9]([-a-z0-9]*)?%s([-a-z0-9]*)?[a-z0-9]|[a-z0-9]([-a-z0-9]*)?%s)(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Examples:

```
publicDomainTemplate: "%s.%s.company.my"
```

```
publicDomainTemplate: "%s.%s.kube.company.my"
```

- **settings.defaultClusterStorageClass**
  string

  The name of the StorageClass that DP will define as the default StorageClass used in the cluster.

  DP prohibits manually adding or modifying the default annotation (`storageclass.kubernetes.io/is-default-class='true'`) for StorageClass. Use the parameter to define or change the StorageClass that is used as the default in the cluster.

  If the parameter is omitted, the default StorageClass is either:

  - A StorageClass in the cluster that has the default annotation (`storageclass.kubernetes.io/is-default-class='true'`);
  - The first (alphabetically) StorageClass created by the cloud provider module (if any cloud provider module is used).

  Use this parameter to change the default StorageClass in the cluster, as manually adding the default annotation to a StorageClass is not allowed.

  Default:

- **settings.highAvailability**
  boolean

  A global switch to enable the *high availability* mode for modules that support it.

  If not defined, the value is determined automatically as `true` for clusters with more than one master node. Otherwise, it is determined as `false`.

  Examples:

  ```
  highAvailability: true
  ```

  ```
  highAvailability: false
  ```

- **settings.moduleV2Enabled**
  boolean

  Enables the experimental Module v2 controller and installs the Module v2 CRD instead of the legacy Module CRD.

  The feature is experimental and not supported under the Deckhouse SLA. Use it with caution.

  Default: `false`

  Example:

  ```
  moduleV2Enabled: false
  ```

- **settings.modules**
  object

  Common parameters of Deckhouse modules.

  - **settings.modules.gatewayAPI**
    object

    Common parameters for Gateway API usage by DP modules.

    - **settings.modules.gatewayAPI.enabled**
      boolean

      Enables the use of Gateway API resources (Gateway, HTTPRoute, etc.) by the DP modules that support it.

      Unlike ingressClass, there is no built-in default Gateway, so Gateway API is only actually used by a module if this flag is enabled **and** a Gateway can be resolved for it — either from this parameter's gateway field, a module-specific override, or a Gateway automatically discovered in the cluster.

      Modules that support Gateway API can override this setting with their own `gatewayAPI.enabled` parameter.

    - **settings.modules.gatewayAPI.gateway**
      object

      Reference to the Gateway object (from the Gateway API) used by DP modules when creating Gateway API resources.

      If not specified, modules use the Gateway automatically discovered in the cluster (if any).

      - **settings.modules.gatewayAPI.gateway.name**
        string

Required value

Name of the Gateway object.

- **settings.modules.gatewayAPI.gateway.namespace**
  string

  Required value

  Namespace of the Gateway object.

- **settings.modules.gatewayAPIGateway**
  Deprecated
  object

  > **Deprecated**. Use settings.modules.gatewayAPI.gateway instead.

  Reference to the Gateway object (from the Gateway API) used by DP modules when creating Gateway API resources.

  If not specified, modules use the Gateway automatically discovered in the cluster (if any).

  - **settings.modules.gatewayAPIGateway.name**
    string

    Required value

    Name of the Gateway object.

  - **settings.modules.gatewayAPIGateway.namespace**
    string

    Required value

    Namespace of the Gateway object.

- **settings.modules.https**
  object

  HTTPS access implementation method used by the Deckhouse ecosystem modules.

  Examples:

  ```
  https:
    certManager:
      clusterIssuerName: letsencrypt
    mode: CertManager
  ```

  ```
  https:
    mode: Disabled
  ```

  ```
  https:
    mode: OnlyInURI
  ```

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: plainstring
  ```

  - **settings.modules.https.certManager**
    object
    - **settings.modules.https.certManager.clusterIssuerName**
      string

      ClusterIssuer name used by modules to obtain TLS certificates.

      The `cert-manager` module offers the following ClusterIssuer options:

      - `letsencrypt`
      - `letsencrypt-staging`
      - `selfsigned`
      - `clouddns`
      - `cloudflare`
      - `digitalocean`
      - `route53`

      You can use your own ClusterIssuer if necessary.

      Default: `letsencrypt`

  - **settings.modules.https.customCertificate**
    object

- **settings.modules.https.customCertificate.secretName**
  string

  Name of the Secret in the `d8-system` namespace containing a TLS certificate for modules.

  This Secret must be in the `kubernetes.io/tls` format.

  Default: `false`

- **settings.modules.https.mode**
  string

  HTTPS usage mode:

  - `CertManager`: Modules use HTTPS and obtain certificates automatically using the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate`: Modules use HTTPS with the certificate from the `d8-system` namespace.
  - `Disabled`: Modules use HTTP only (some modules, such as `user-authn`, may not work).
  - `OnlyInURI`: Modules use HTTP in the expectation that there is an external upstream HTTPS load balancer, which terminates HTTPS. This load balancer should be capable of redirecting requests from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.modules.ingress**
  object

  Common parameters for Ingress usage by DP modules.

  - **settings.modules.ingress.enabled**
    boolean

    Enables the creation of Ingress resources by the DP modules that support it.

    Modules that support Ingress can override this setting with their own `ingress.enabled` parameter.

    Default: `true`

  - **settings.modules.ingress.ingressClass**
    string

    Default IngressClass name used by DP modules when creating Ingress resources.

    Some modules support their own `ingressClass` parameter, which lets you use a different IngressClass instead of the default one.

    Default: `nginx`

- **settings.modules.ingressClass**
  Deprecated
  string

  > **i** **Deprecated**. Use settings.modules.ingress.ingressClass instead.

  Default IngressClass name used by DP modules when creating Ingress resources.

  Some modules support their own `ingressClass` parameter, which lets you use a different IngressClass instead of the default one.

  Default: `nginx`

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: nginx
  ```

- **settings.modules.placement**
  object

  Parameters regulating the layout of Deckhouse module components.

  - **settings.modules.placement.customTolerationKeys**
    array of strings

    A list of custom toleration keys; use them to allow the deployment of some critical add-ons (such as cni and csi) on dedicated nodes.

    Example:

    ```
    customTolerationKeys:
    - dedicated.example.com
    - node-dedicated.example.com/master
    ```

- **settings.modules.publicDomainTemplate**
  string

  A DNS name template with the `%s` key used as a dynamic part of the string.

  Deckhouse modules use this template when creating Ingress resources.

  For example, given the template `%s.kube.company.my`, the `prometheus` module will create an Ingress resource at the address `grafana.kube.company.my` to provide access to Grafana.

  Pay attention to the following:

  - If this parameter is not specified, Ingress resources will not be created.
  - The domain specified in the template must not match or be a subdomain of the domain specified in the `network.clusterDomain` parameter of the `control-plane-manager` module. It is not recommended to change the cluster domain unless absolutely necessary.
  - For the template to function correctly, you must first configure DNS services both in the networks where the cluster nodes will be located and in the networks from which clients will access the platform's service web interfaces.
  - If the template matches the domain of the node network, use only A records to assign addresses of the nodes' Frontend interfaces to the platform's service web interfaces. For example, if the nodes are registered under the `company.my` zone and the template is `%s.company.my`.

  Pattern: `^(%s([-a-z0-9]*[a-z0-9])?|[a-z0-9]([-a-z0-9]*)?%s([-a-z0-9]*)?[a-z0-9]|[a-z0-9]([-a-z0-9]*)?%s)(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Examples:

  ```
  publicDomainTemplate: "%s.kube.company.my"
  ```

  ```
  publicDomainTemplate: kube-%s.company.my
  ```

- **settings.modules.resourcesRequests**
  object

  The amount of resources (CPU and memory) allocated to some components.

  More info in the Kubernetes documentation.

  - **settings.modules.resourcesRequests.controlPlane**
    Deprecated
    object

    > **i** **Deprecated**. Use `control-plane-manager` moduleconfig parameter `.spec.settings.resourcesRequests` instead.

    Amount of resources (CPU and memory) allocated to control-plane components on each master node.

    These settings do not apply if the cluster control plane is managed by a cloud provider (for example, Google Kubernetes Engine or Azure Kubernetes Service).

    Example:

    ```
    controlPlane:
      cpu: 1000m
      memory: 500M
    ```

    - **settings.modules.resourcesRequests.controlPlane.cpu**
      Deprecated
      string or number

      > **i** **Deprecated**. Use `controlPlaneManager.spec.settings.resourcesRequests.cpu` instead.

      The combined CPU requests for control-plane components on each master node.

      Pattern: `^[0-9]+m?$`

    - **settings.modules.resourcesRequests.controlPlane.memory**
      Deprecated
      string

      > **i** **Deprecated**. Use `controlPlaneManager.spec.settings.resourcesRequests.memory` instead.

      The combined memory requests for control-plane components on each master node.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.modules.storageClass**
  string

  The StorageClass to use with all Deckhouse Platform components (Prometheus, Grafana, OpenVPN, etc.) unless it is overridden in the module configuration.

  If the value is not specified, the default StorageClass set in the cluster is used.

  Important notes:

  - If the module's configuration defines its own `storageClass` parameter, the module's value takes precedence and the global `modules.storageClass` is not used.
  - If a PersistentVolumeClaim (PVC) for a module already exists, changing this parameter does not affect the existing PVC or require disk recreation. The module uses the existing PVC and its StorageClass. See the prometheus module example for more details on how this parameter is applied.
  - This parameter is applied during module activation when creating new PVCs.
  - Use this parameter only in exceptional circumstances.

  Default:

- **settings.prefix**
  string

  A prefix of the objects to be created in the cloud.

  Is used, for example, to distinguish objects created for different clusters, to configure routing, etc.

  This is the new home for the prefix that was previously set in `ClusterConfiguration.cloud.prefix`. If not set, the value from `ClusterConfiguration.cloud.prefix` is used.

  The prefix is used to name cloud objects such as disks (e.g. `<prefix>-kubernetes-data-0`). Cloud object names are limited to 63 characters, so the prefix is capped at 45 characters to leave room for the generated suffix. Depending on the provider and the number of master nodes, the effective limit may be lower; this is additionally enforced by a bootstrap preflight check.

  > **Warning.** This value names existing cloud infrastructure (security groups, route tables, resource groups, instances, etc.). It must match the prefix the cluster was bootstrapped with — setting a different value will break cloud integration. It is not intended to be changed in a running cluster.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Maximum length: `45`

  Example:

  ```
  prefix: kube
  ```

- **settings.storageClass**
  Deprecated
  string

  > **Deprecated**. Use settings.modules.storageClass instead.

  The StorageClass to use with all Deckhouse Platform components (Prometheus, Grafana, OpenVPN, etc.).

  - If the value is not specified, the default StorageClass set in the cluster is used.
  - Use this parameter only in exceptional circumstances.
  - This parameter is applied during module activation.

  Default:

Additional resources

- Module `control-plane-manager` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

Application

Scope: Namespaced
Version: v1alpha1

Application defines an installed application instance in a namespace.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: example
spec:
  maintenance: NoResourceReconciliation
  packageName: console
  packageRepositoryName: deckhouse
  packageVersion: v1.0.0
  releaseChannel: stable
```

- **spec**
  object

  Required value

  Application configuration.

  - **spec.maintenance**
    string

    Application maintenance mode.

    Supported modes:

    - `NoResourceReconciliation`: A mode for developing or debugging the application.

      In this mode:

      - Configuration or hook changes are not reconciled, which prevents resources from being updated automatically.
      - Resource monitoring is disabled, which prevents deleted resources from being restored.
      - All the application's resources are labeled with `maintenance.deckhouse.io/no-resource-reconciliation`.
      - The `ApplicationIsInMaintenanceMode` alert is triggered.

    Allowed values: `NoResourceReconciliation`

    Example:

    ```
    maintenance: NoResourceReconciliation
    ```

  - **spec.packageName**
    string

    Required value

    Name of the application package to install.

    Example:

    ```
    packageName: console
    ```

  - **spec.packageRepositoryName**
    string

    Name of the PackageRepository object where the application package is located.

    Example:

    ```
    packageRepositoryName: deckhouse
    ```

  - **spec.packageVersion**
    string

    Required value

    Version of the application package to install.

    Example:

    ```
    packageVersion: v1.0.0
    ```

  - **spec.resourceRequests**
    array of objects

    Resource settings, replica counts, and disk sizes for workloads deployed by the application.

    Specified values override the settings provided by the application package. Workloads, containers, and disks not listed in this parameter retain their settings from the package.

    Each workload is identified by a combination of kind (`kind`) and name (`name`). Only one entry is allowed for

each combination.

Resource settings, replica count, and disk sizes for a single application workload.

- **spec.resourceRequests.containers**
  array of objects

  Compute resources of the workload’s containers.

  If the specified container does not exist in the workload, the entry is ignored.

  Resource settings for a single workload container.

  - **spec.resourceRequests.containers.name**
    string

    Required value

    Name of the container inside the workload.

    Example:

    ```
    name: controller
    ```

  - **spec.resourceRequests.containers.resources**
    object

    Container resource requests and limits, or the range within which VerticalPodAutoscaler sets the container’s requests.

    Requests and limits use the same format as the `resources` field of a Pod container. Supports `cpu`, `memory`, and `ephemeral-storage`.

    Specified values are merged with the settings from the application package separately for each resource. For example, if only `memory` is specified, the `cpu` value from the package is retained.

    Resource requests or limits defined in the package cannot be removed. Their values can only be overridden.

    If VerticalPodAutoscaler controls the container’s `cpu` or `memory`, a request set for that resource remains fixed. If a `cpu` or `memory` limit is set, VPA does not change the container’s limits. The `ApplicationVerticalPodAutoscalerPinned` alert is triggered if a request or limit is set for a resource controlled by VPA.

    To let VPA choose requests within a range, set `min` and `max` instead of `cpu` and `memory` requests and limits.

    - **spec.resourceRequests.containers.resources.claims**
      array of objects

      List of resource claims used by the container.

      Each entry references a resource claim defined in `pod.spec.resourceClaims`. Requires the `DynamicResourceAllocation` feature gate to be enabled.

      This field is immutable. It can only be set for containers.

      Resource claim used by the container.

      - **spec.resourceRequests.containers.resources.claims.name**
        string

        Required value

        Name of a ResourceClaim defined in `pod.spec.resourceClaims`.

        The referenced claim makes its resources available to the container.

      - **spec.resourceRequests.containers.resources.claims.request**
        string

        Name of a request defined in the referenced resource claim.

        If specified, only the resources allocated for this request are made available to the container. If omitted, all resources allocated for the claim are made available.

    - **spec.resourceRequests.containers.resources.limits**
      object

      Maximum amount of compute resources available to the container.

    - **spec.resourceRequests.containers.resources.max**
      object

      Upper bound of the container’s `cpu` and `memory` requests that VerticalPodAutoscaler sets (`maxAllowed` in its policy for the container).

      Set together with `min` for the same resources. Within the range, the requests are chosen by VPA, so a container with `min` and `max` cannot have `cpu` or `memory` requests or limits. A resource bound cannot be less than in `min`.

If no VerticalPodAutoscaler of the application sets a resource of the container, for example, because the `vertical-pod-autoscaler` module is disabled, the container gets `min` as the request and `max` as the limit of the resource, and the `ApplicationVerticalPodAutoscalerMissing` alert is triggered.

- **spec.resourceRequests.containers.resources.max.cpu**
  integer or string

  Bound of the `cpu` request.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  cpu: 250m
  ```

- **spec.resourceRequests.containers.resources.max.memory**
  integer or string

  Bound of the `memory` request.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  memory: 256Mi
  ```

- **spec.resourceRequests.containers.resources.min**
  object

  Lower bound of the container's `cpu` and `memory` requests that VerticalPodAutoscaler sets (`minAllowed` in its policy for the container).

  Set together with `max` for the same resources. Within the range, the requests are chosen by VPA, so a container with `min` and `max` cannot have `cpu` or `memory` requests or limits. To fix a resource at one value, specify the same value in `min` and `max`.

  If no VerticalPodAutoscaler of the application sets a resource of the container, for example, because the `vertical-pod-autoscaler` module is disabled, the container gets `min` as the request and `max` as the limit of the resource, and the `ApplicationVerticalPodAutoscalerMissing` alert is triggered.

  - **spec.resourceRequests.containers.resources.min.cpu**
    integer or string

    Bound of the `cpu` request.

    Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    Example:

    ```
    cpu: 250m
    ```

  - **spec.resourceRequests.containers.resources.min.memory**
    integer or string

    Bound of the `memory` request.

    Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    Example:

    ```
    memory: 256Mi
    ```

- **spec.resourceRequests.containers.resources.requests**
  object

  Minimum amount of compute resources required by the container.

  If this parameter is not specified, the `limits` value is used if explicitly set. Otherwise, the value is implementation-defined.

  The `requests` value cannot exceed `limits`.

- **spec.resourceRequests.kind**
  string

  Required value

  Kind of workload for which resource settings are specified.

  Allowed values: `Deployment`, `StatefulSet`, `DaemonSet`, `ReplicaSet`, `Pod`, `Job`, `CronJob`

Example:

```
kind: Deployment
```

- **spec.resourceRequests.maxReplicas**
  integer

  Maximum number of workload replicas for the HorizontalPodAutoscaler that the application deploys for the workload.

  Applies only to Deployment, StatefulSet, and ReplicaSet. Set together with `minReplicas`. Cannot be set together with `replicas`: within the range, the number of replicas is chosen by the HorizontalPodAutoscaler.

  If the application does not deploy a HorizontalPodAutoscaler for the workload, the workload runs `minReplicas` replicas, and the `ApplicationHorizontalPodAutoscalerMissing` alert is triggered.

  Allowed values: `1 <= X`

  Example:

  ```
  maxReplicas: 10
  ```

- **spec.resourceRequests.minReplicas**
  integer

  Minimum number of workload replicas for the HorizontalPodAutoscaler that the application deploys for the workload.

  Applies only to Deployment, StatefulSet, and ReplicaSet. Set together with `maxReplicas`. Cannot be set together with `replicas`: within the range, the number of replicas is chosen by the HorizontalPodAutoscaler. Cannot be greater than `maxReplicas`.

  If the application does not deploy a HorizontalPodAutoscaler for the workload, the workload runs `minReplicas` replicas, and the `ApplicationHorizontalPodAutoscalerMissing` alert is triggered.

  Allowed values: `1 <= X`

  Example:

  ```
  minReplicas: 2
  ```

- **spec.resourceRequests.name**
  string

  Required value

  Workload name as specified in the application package.

  Example:

  ```
  name: controller
  ```

- **spec.resourceRequests.replicas**
  integer

  Number of workload replicas.

  Applies only to workloads that support replica management: Deployment, StatefulSet, and ReplicaSet. Cannot be set together with `minReplicas` or `maxReplicas`.

  If the application deploys a HorizontalPodAutoscaler for the workload, its `minReplicas` and `maxReplicas` are set to this value, so the number of replicas does not change, and the `ApplicationHorizontalPodAutoscalerPinned` alert is triggered. To let HorizontalPodAutoscaler scale the workload, set `minReplicas` and `maxReplicas` instead. The value `0` applies only to the workload: HorizontalPodAutoscaler does not scale a workload with no replicas.

  Allowed values: `0 <= X`

  Example:

  ```
  replicas: 3
  ```

- **spec.resourceRequests.volumeClaimTemplates**
  array of objects

  Disk sizes for the StatefulSet's volume claim templates.

  Applies only to StatefulSet. Each template is identified by its name, the `metadata.name` of an entry in `spec.volumeClaimTemplates`. If the specified template does not exist in the StatefulSet, the entry is ignored.

  The size cannot be less than the size of the existing disks: a disk can only be expanded. Expanding requires the StorageClass of the disks to allow volume expansion (`allowVolumeExpansion: true`).

  Disk size for a single volume claim template of a StatefulSet.

- **spec.resourceRequests.volumeClaimTemplates.name**
  string

  Required value

  Name of the volume claim template in the StatefulSet.

  Example:

  ```
  name: data
  ```

- **spec.resourceRequests.volumeClaimTemplates.storage**
  integer or string

  Required value

  Disk size requested from the template for each StatefulSet replica.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  storage: 10Gi
  ```

- **spec.settings**
  object

  Configuration application settings.

- **spec.settingsVersion**
  number

  Version of the settings schema the settings are written for. Distinct from packageVersion, which selects the application package.

  Settings of an older version are converted to the latest one the package provides. If not specified, version 1 is used.

- **spec.update**
  object

  Application update settings: update mode and update windows.

  If this parameter is not specified, the `Manual` update mode is used.

  Update notifications cannot be configured for an application: the `update.notification` settings of the `deckhouse` module are used.

  Example:

  ```
  update:
    mode: Auto
    windows:
    - from: '8:00'
      to: '15:00'
      days:
      - Tue
      - Sat
  ```

  - **spec.update.mode**
    string

    Update mode of the application on its release channel.

    - `AutoPatch`: Automatic update mode for patch versions.

      To change a minor version (for example, from `v1.2.*` to `v1.3.*`), confirmation is required.

      A patch version update (for example, from `v1.3.1` to `v1.3.2`) is applied taking into account the update windows, if they are set.

    - `Auto`: Automatic update mode for all versions.

      Minor version updates (for example, from `v1.2.*` to `v1.3.*`) and patch version updates (for example, from `v1.3.1` to `v1.3.2`) are applied taking into account the update windows, if they are set.

    - `Manual`: Manual update mode for all versions.

      Confirmation is required for updating both minor and patch versions.

    Allowed values: `AutoPatch`, `Auto`, `Manual`

    Examples:

```
mode: AutoPatch
```

```
mode: Auto
```

```
mode: Manual
```

- **spec.update.windows**
  array of objects

  List of application update windows.

  Time interval during the day when application updates are allowed.

  - **spec.update.windows.days**
    array of strings

    Days of the week on which the update window is applied.

    If the parameter is omitted, the update window is applied every day.

    - **Element of the array**
      string

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

  - **spec.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be earlier than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

    Should be later than the start time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

- **status**
  object

  Current state of the application.

  - **status.conditions**
    array of objects

    Conditions reflecting the current state of the application.

    Condition reflecting an aspect of the current state of the application.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition status last changed.

    - **status.conditions.message**
      string

      Required value

      Message with additional information about the condition.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Value of `.metadata.generation` for which the condition was set.

Indicates whether the condition corresponds to the current generation of the instance.

Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  Programmatic identifier indicating the reason for the last condition status change.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type in CamelCase or in the format `foo.example.com/CamelCase`.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.currentVersion**
  object

  Information about the currently installed application version.

  - **status.currentVersion.version**
    string

    Installed application version in SemVer format.

- **status.lastAppliedConfiguration**
  object

  Application configuration used during the last successful apply.

  Includes user-defined settings and default values defined in the configuration schema.

- **status.summary**
  object

  Summary of the application state, including messages and recommendations for the user.

  The controller updates this information during each reconciliation. Each application has exactly one lifecycle state.

  Use this information to display the application state in the web interface. Do not recalculate the state from conditions.

  - **status.summary.message**
    string

    Description of the current application state.

  - **status.summary.state**
    string

    Current application lifecycle state.

    Examples:

    ```
    state: Pending
    ```

    ```
    state: Failed
    ```

    ```
    state: Updating
    ```

    ```
    state: Configuring
    ```

    ```
    state: Ready
    ```

```
state: Degraded
```

```
state: Suspended
```

```
state: Deleting
```

- **status.summary.tip**
  string

  Recommended action for the current application state.

  May be empty if no additional action is required.

- **status.tracking**
  object

  Nelm report on application status.

- **status.update**
  object

  Update of the application: the update plan, the version the automatic update moves to next and the notification sent about the pending update.

  Empty before the first version is installed.

  - **status.update.automaticVersion**
    string

    Version the automatic update installs next. It changes only after the previous one is installed and ready. Empty in the `Manual` update mode.

  - **status.update.notified**
    object

    Notification sent about the pending update: the version it announced and when it was sent.

    It is sent to the webhook of the `update.notification` settings of the `deckhouse` module. The update is installed no earlier than `minimalNotificationTime` after the notification. Cleared once a new version is installed or the update is cancelled.

    - **status.update.notified.time**
      string

      Required value

      Time the notification was sent.

    - **status.update.notified.version**
      string

      Required value

      Version the notification announced.

  - **status.update.plan**
    array of strings

    Update path after the automatic version in every update mode. The path follows the compatibility rules to the newest version in the cluster, or to the newest version the rules reach; each step is the newest version from which the end of the path is still reached.

- **status.urls**
  array of objects

  Application endpoint URLs obtained from Ingress, HTTPRoute, and ServiceAccount resources in the application chart with the `packages.deckhouse.io/application-endpoint-description` annotation. A ServiceAccount gives the in-cluster address of the Kubernetes API, `https://kubernetes.default.svc/`.

  Application endpoint defined by an Ingress, HTTPRoute, or ServiceAccount resource in the application chart.

  - **status.urls.description**
    string

    Endpoint description from the `packages.deckhouse.io/application-endpoint-description` annotation.

    The description is empty if the annotation value is `"true"`.

  - **status.urls.url**
    string

    Required value

    Application endpoint URL.

ApplicationPackage

Short names: ap

Scope: Cluster
Version: v1alpha1

ApplicationPackage defines an application package available for installation.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ApplicationPackage
metadata:
  name: example
```

- **status**
  object

  Current state of the application package.

  - **status.availableRepositories**
    array of strings

    List of repository names where the application package is available.

  - **status.releaseChannels**
    object

    Application package versions associated with release channels in each repository.

    The first-level key is the package repository name, and the second-level key is the release channel name.

  - **status.usedBy**
    array of objects

    List of applications installed from this package.

    Application installed from the package.

    - **status.usedBy.name**
      string

      Name of the application instance.

    - **status.usedBy.namespace**
      string

      Namespace where the application is installed.

    - **status.usedBy.repositoryName**
      string

      Name of the PackageRepository from which the application was installed.

    - **status.usedBy.version**
      string

      Package version used by the application.

  - **status.usedByCount**
    integer

    Number of applications installed from this package.

    Equals the number of entries in `usedBy`.

ApplicationPackageVersion

Short names: apv

Scope: Cluster
Version: v1alpha1

ApplicationPackageVersion defines a version of an application package.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ApplicationPackageVersion
metadata:
  name: example
spec:
  packageName: console
  packageRepositoryName: deckhouse
  packageVersion: v1.0.0
```

- **spec**

object

Application package version parameters.

- **spec.packageName**
  string

  Required value

  Name of the application package.

  Example:

  ```
  packageName: console
  ```

- **spec.packageRepositoryName**
  string

  Required value

  Name of the package repository containing the package.

  Example:

  ```
  packageRepositoryName: deckhouse
  ```

- **spec.packageVersion**
  string

  Required value

  Version of the application package.

  Example:

  ```
  packageVersion: v1.0.0
  ```

- **status**
  object

  Current state of the application package version.

  - **status.conditions**
    array of objects

    Conditions reflecting the current state of the application package version.

    Condition reflecting an aspect of the current state of the package version.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition status last changed.

    - **status.conditions.message**
      string

      Required value

      Message with additional information about the condition.

      May be empty.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Value of `.metadata.generation` for which the condition was set.

      Indicates whether the condition corresponds to the current generation of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      Programmatic identifier indicating the reason for the last condition status change.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

Required value

Condition status.

Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type in CamelCase or in the format `foo.example.com/CamelCase`.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.observedGeneration**
  integer

  Value of `.metadata.generation` for which the current status was calculated.

  If the value is lower than the current `.metadata.generation`, the status does not yet reflect the latest changes to the instance.

  Allowed values: `0 <= X`

- **status.packageMetadata**
  object

  Application package version metadata, including the description, requirements, disable options, and version compatibility rules.

  - **status.packageMetadata.category**
    string

    Application package category.

  - **status.packageMetadata.changelog**
    object

    Information about changes in this package version.

    - **status.packageMetadata.changelog.features**
      array of strings

      List of new features in this package version.

    - **status.packageMetadata.changelog.fixes**
      array of strings

      List of bugs fixed in this package version.

  - **status.packageMetadata.description**
    object

    Localized descriptions of the application package.

    - **status.packageMetadata.description.en**
      string

      English description of the application package.

    - **status.packageMetadata.description.ru**
      string

      Russian description of the application package.

  - **status.packageMetadata.disableOptions**
    object

    Parameters of package disable protection.

    - **status.packageMetadata.disableOptions.confirmation**
      boolean

      Indicates whether confirmation is required to disable the application package.

    - **status.packageMetadata.disableOptions.messages**
      object

      Localized messages displayed when confirming package disabling.

      - **status.packageMetadata.disableOptions.messages.en**
        string

        Disable confirmation message in English.

      - **status.packageMetadata.disableOptions.messages.ru**
        string

Disable confirmation message in Russian.

- **status.packageMetadata.licensing**
  object

  Application package licensing information for different editions.

  - **status.packageMetadata.licensing.editions**
    object

    Licensing settings for different application package editions.

    - **status.packageMetadata.licensing.editions.<KEY_NAME>**
      object

      `<KEY_NAME>` — item (key) name.

      - **status.packageMetadata.licensing.editions.<KEY_NAME>.available**
        boolean

        Indicates whether the edition is available for use.

      - **status.packageMetadata.licensing.editions.<KEY_NAME>.enabledInBundles**
        array of strings

        Bundles in which the package is enabled by default in this edition.

        Not used for application packages.

- **status.packageMetadata.requirements**
  object

  System requirements for installing and running the application.

  - **status.packageMetadata.requirements.deckhouse**
    object

    Required Deckhouse Platform version.

    - **status.packageMetadata.requirements.deckhouse.constraint**
      string

      Deckhouse Platform version constraint in SemVer format.

  - **status.packageMetadata.requirements.kubernetes**
    object

    Required Kubernetes version.

    - **status.packageMetadata.requirements.kubernetes.constraint**
      string

      Kubernetes version constraint in SemVer format.

  - **status.packageMetadata.requirements.modules**
    object

    Deckhouse Platform module requirements: required and optional dependencies, groups of alternative modules, and groups of incompatible modules.

    - **status.packageMetadata.requirements.modules.anyOf**
      array of objects

      Groups of alternative modules.

      At least one module in each group must be enabled. If a version constraint is specified for a module, its version must satisfy that constraint.

      These groups do not add edges to the dependency graph.

      Group of alternative modules, at least one of which must be enabled.

      - **status.packageMetadata.requirements.modules.anyOf.description**
        string

        Human-readable description of the group’s purpose.

      - **status.packageMetadata.requirements.modules.anyOf.modules**
        array of objects

        Required value

        Alternative modules in the group.

        Module with an optional version constraint in SemVer format.

        If no constraint is specified, any version is allowed.

        - **status.packageMetadata.requirements.modules.anyOf.modules.constraint**
          string

          Module version constraint in SemVer format.

- **status.packageMetadata.requirements.modules.anyOf.modules.name**
  string

  Required value

  Module name.

- **status.packageMetadata.requirements.modules.anyOf.name**
  string

  Required value

  Stable identifier used by the scheduler in diagnostics.

- **status.packageMetadata.requirements.modules.conditional**
  array of objects

  Optional modules with version constraints.

  The module does not have to be enabled. If it is enabled, its version must satisfy the specified constraint.

  Replaces the deprecated `!optional` suffix from the v1 requirements format.

  Optional module with a version constraint in SemVer format.

  If no constraint is specified, any version is allowed.

  - **status.packageMetadata.requirements.modules.conditional.constraint**
    string

    Module version constraint in SemVer format.

  - **status.packageMetadata.requirements.modules.conditional.name**
    string

    Required value

    Module name.

- **status.packageMetadata.requirements.modules.mandatory**
  array of objects

  Modules required for the application to run.

  Each specified module must be enabled. If a version constraint is specified, the module version must satisfy it.

  Required module with an optional version constraint in SemVer format.

  If no constraint is specified, any version is allowed.

  - **status.packageMetadata.requirements.modules.mandatory.constraint**
    string

    Module version constraint in SemVer format.

  - **status.packageMetadata.requirements.modules.mandatory.name**
    string

    Required value

    Module name.

- **status.packageMetadata.requirements.modules.noneOf**
  array of objects

  Groups of incompatible modules.

  For the application to run, none of the modules in these groups may be enabled with an incompatible version.

  If no version constraint is specified, all versions of the module are considered incompatible. If a constraint is specified, only versions satisfying that constraint are considered incompatible.

  These groups do not add edges to the dependency graph.

  Group of incompatible modules.

  None of the listed modules may be enabled with an incompatible version.

  - **status.packageMetadata.requirements.modules.noneOf.description**
    string

    Human-readable description of the group’s purpose.

  - **status.packageMetadata.requirements.modules.noneOf.modules**
    array of objects

    Required value

    Incompatible modules in the group.

Module with an optional version constraint in SemVer format.

If no constraint is specified, all versions of the module are considered incompatible.

- **status.packageMetadata.requirements.modules.noneOf.modules.constraint**
  string

  Module version constraint in SemVer format defining the range of incompatible versions.

- **status.packageMetadata.requirements.modules.noneOf.modules.name**
  string

  Required value

  Module name.

- **status.packageMetadata.requirements.modules.noneOf.name**
  string

  Required value

  Stable identifier used by the scheduler in diagnostics.

- **status.packageMetadata.stage**
  string

  Life cycle stage of the application package.

  Examples:

  ```
  stage: Experimental
  ```

  ```
  stage: Preview
  ```

  ```
  stage: General Availability
  ```

  ```
  stage: Deprecated
  ```

- **status.packageMetadata.versionCompatibilityRules**
  object

  Version compatibility rules for upgrades and downgrades.

  - **status.packageMetadata.versionCompatibilityRules.downgrade**
    object

    Compatibility rules for downgrading from this application version.

    Default: `{"step":{"major":0,"minor":0,"patches":-1}}`

    - **status.packageMetadata.versionCompatibilityRules.downgrade.step**
      object

      Compatibility rule step.

      - **status.packageMetadata.versionCompatibilityRules.downgrade.step.major**
        integer

        Number of major versions that can be skipped during a downgrade.

        The `-1` value removes all limits, `0` forbids the change.

        Defaults to `0`.

        Allowed values: `-1 <= X`

      - **status.packageMetadata.versionCompatibilityRules.downgrade.step.minor**
        integer

        Number of minor versions that can be skipped during a downgrade.

        The `-1` value removes all limits, `0` forbids the change.

        Defaults to `0`.

        Allowed values: `-1 <= X`

      - **status.packageMetadata.versionCompatibilityRules.downgrade.step.patches**
        integer

        Number of patch versions that can be skipped during a downgrade.

        The `-1` value removes all limits, `0` forbids the change.

        Defaults to `-1`.

        Allowed values: `-1 <= X`

- **status.packageMetadata.versionCompatibilityRules.upgrade**
  object

  Compatibility rules for upgrading to and from this application version.

  Default: `{"step":{"major":0,"minor":1,"patches":-1}}`

  - **status.packageMetadata.versionCompatibilityRules.upgrade.from**
    string

    SemVer constraint on the application versions from which the application can be upgraded to this version.

    If specified, the constraint takes priority over the `step` limits of the version being upgraded from: the application is upgraded to this version only from the versions that satisfy the constraint, regardless of those limits. This applies to the update plan, the automatic update, and a change of `spec.packageVersion` of the Application. An invalid constraint forbids upgrading to this version.

    If not specified, the `step` limits of the version being upgraded from apply.

  - **status.packageMetadata.versionCompatibilityRules.upgrade.step**
    object

    Compatibility rule step.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.step.major**
      integer

      Number of major versions that can be skipped during an upgrade.

      The `-1` value removes all limits, `0` forbids the change.

      Defaults to `0`.

      Allowed values: `-1 <= X`

    - **status.packageMetadata.versionCompatibilityRules.upgrade.step.minor**
      integer

      Number of minor versions that can be skipped during an upgrade.

      The `-1` value removes all limits, `0` forbids the change.

      Defaults to `1`.

      Allowed values: `-1 <= X`

    - **status.packageMetadata.versionCompatibilityRules.upgrade.step.patches**
      integer

      Number of patch versions that can be skipped during an upgrade.

      The `-1` value removes all limits, `0` forbids the change.

      Defaults to `-1`.

      Allowed values: `-1 <= X`

- **status.packageSchemas**
  object

  OpenAPI schemas for validating user-defined application settings and final values passed to package hooks and templates.

  - **status.packageSchemas.settingsConversions**
    array of objects

    Conversions that bring settings written for an older settings schema version to the latest one, ordered by version.

    Empty when the package has a single settings schema version.

    PackageSettingsConversion converts settings of the previous settings schema version to Version.

    - **status.packageSchemas.settingsConversions.descriptions**
      object

      Localized description of the changes in this settings schema version.

      - **status.packageSchemas.settingsConversions.descriptions.en**
        string

        English description of the conversion.

      - **status.packageSchemas.settingsConversions.descriptions.ru**
        string

        Russian description of the conversion.

    - **status.packageSchemas.settingsConversions.expr**
      array of strings

Required value

jq expressions applied to the settings in order.

- **status.packageSchemas.settingsConversions.version**
  integer

  Required value

  Settings schema version the conversion produces; it converts settings of the previous version.

- **status.packageSchemas.settingsSchema**
  object

  OpenAPI v3 schema for validating user-defined application settings in `Application.spec.settings`.

  The schema is stored as an opaque object because it has a recursive JSON Schema structure. The controller validates it when loading the package metadata.

- **status.packageSchemas.valuesSchema**
  object

  OpenAPI v3 schema for validating the final set of `values` passed to package hooks and templates.

  The final set is formed by merging default values with user-defined application settings.

  The schema is stored as an opaque object because it has a recursive JSON Schema structure. The controller validates it when loading the package metadata.

- **status.usedBy**
  array of objects

  List of applications installed from this package version.

  Application installed from the package version.

  - **status.usedBy.name**
    string

    Name of the application instance.

  - **status.usedBy.namespace**
    string

    Namespace where the application is installed.

- **status.usedByCount**
  integer

  Number of applications installed from this package version.

  Equals the number of entries in `usedBy`.

  Allowed values: `0 <= X`

ClusterConfiguration

Version: deckhouse.io/v1

General parameters of a cluster.

Defines, for example, network and CRI parameters. Some parameters can be changed after the cluster is bootstrapped, during its operation.

To change the `ClusterConfiguration` resource in a running cluster, run the following command:

```
d8 system edit cluster-configuration
```

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
podSubnetNodeCIDRPrefix: '24'
podSubnetCIDR: 10.244.0.0/16
serviceSubnetCIDR: 192.168.0.0/16
clusterDomain: k8s.internal
clusterType: Cloud
cloud:
  prefix: k8s-dev
  provider: Yandex
proxy:
  httpProxy: https://user:password@proxy.company.my:8443
  httpsProxy: https://user:password@proxy.company.my:8443
  noProxy:
  - company.my
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **cloud**
  object

  Cloud provider-related settings (if the `Cloud` clusterType is used).

  - **cloud.prefix**
    Deprecated
    string

    A prefix of the objects to be created in the cloud.

    Deprecated. Use `prefix` in the `global` ModuleConfig instead.

    This field is still used by the installer (dhctl) to name cloud infrastructure at bootstrap and must remain set for cloud clusters; the ModuleConfig value is the source for in-cluster consumers and must match it.

    Is used, for example, to distinguish objects created for different clusters, to configure routing, etc.

    Cloud object names (e.g. disks named `<prefix>-kubernetes-data-0`) are limited to 63 characters, so the prefix is capped at 45 characters. The effective limit may be lower depending on the provider and the number of master nodes; this is additionally enforced by a bootstrap preflight check.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `45`

  - **cloud.provider**
    string

    Required value

    Cloud provider.

    Allowed values: `OpenStack`, `AWS`, `GCP`, `Yandex`, `vSphere`, `Azure`, `VCD`, `Zvirt`, `Dynamix`, `Huaweicloud`, `DVP`

- **clusterDomain**
  Deprecated
  string

  Deprecated. Use `clusterDomain` in the `network` section of the `control-plane-manager` ModuleConfig instead.

  Cluster domain (used for local routing).

  Please note: the domain must not match the domain used in the DNS name template in the publicDomainTemplate parameter. For example, you cannot set `clusterDomain: cluster.local` and `publicDomainTemplate: %s.cluster.local` at the same time.

  > **i** If you need to change a parameter in a running cluster, it is recommended to use instructions

  The field still works, but it will be removed in a future release. Until the value is moved to the ModuleConfig, the cluster reports the `D8ObsoleteClusterDomainInClusterConfiguration` alert. Remove it here only after the ModuleConfig carries the domain.

- **clusterType**
  string

  Required value

  Type of the cluster infrastructure:

  - `Static` — a cluster on bare metal (physical servers) or virtual machines. In the case of virtual machines, it is assumed that Deckhouse doesn't have access to the API for managing virtual machines (they are managed by the administrator using the usual cloud infrastructure tools);
  - `Cloud` — a cluster deployed on the resources of a cloud infrastructure. This type implies that Deckhouse has access to the cloud infrastructure API for managing virtual machines.

  Allowed values: `Cloud`, `Static`

- **defaultCRI**
  string

  The container runtime type that used on cluster nodes (NodeGroups) by default.

  If the value `NotManaged` is used, then Deckhouse does not manage the container runtime (and doesn't install it). In this case, it is necessary to use images for NodeGroups on which the container runtime is already installed.

  If `ContainerdV2` is set, `CgroupsV2` will be used (providing improved security and resource management). To use `ContainerdV2` as the container runtime, cluster nodes must meet the following requirements:

- Support for `CgroupsV2`.
- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- Systemd version `244` or newer.
- Support for `erofs` kernel module.

Default: `Containerd`

Allowed values: `Containerd`, `ContainerdV2`, `NotManaged`

- **encryptionAlgorithm**
  Deprecated
  string

  Deprecated. Use `encryptionAlgorithm` in the `control-plane-manager` ModuleConfig instead.

  When the ModuleConfig field is set, it takes precedence over this field.

  Starting from version **1.31**, kubeadm use the specified asymmetric encryption algorithm when generating keys and certificates for the following control-plane components:

  - `apiserver`
  - `apiserver-kubelet-client`
  - `apiserver-etcd-client`
  - `front-proxy-client`
  - `etcd-server`
  - `etcd-peer`
  - `etcd-healthcheck-client`

  Certificates for the components listed above will be reissued using the selected algorithm and key length.

  > **Warning.** When reissuing certificates, the root certificate (**CA**) is not rotated. The root certificate is created with the selected algorithm only during the initial cluster bootstrap.

  Allowed values: `RSA-2048`, `RSA-3072`, `RSA-4096`, `ECDSA-P256`

- **kind**
  string

  Required value

  Allowed values: `ClusterConfiguration`

- **kubernetesVersion**
  Deprecated
  string

  Deprecated. Use `kubernetesVersion` in the `control-plane-manager` ModuleConfig instead (`d8 k edit mc control-plane-manager`).

  The cluster Kubernetes version is resolved in the following order: `kubernetesVersion` parameter in the `control-plane-manager` ModuleConfig, then this field, then the version the current DP release defaults to. The ModuleConfig setting wins whenever it is set, `Default` included.

  This field is optional. While it pins a version and the `kubernetesVersion` in the `control-plane-manager` ModuleConfig setting is unset, this field is what decides the cluster version. The `D8ObsoleteKubernetesVersionFieldInClusterConfiguration` alert appears in the cluster simply because the field is present — including when the ModuleConfig setting already decides the version — and clears it once the field is removed.

  Do not change this field in a running cluster to upgrade or downgrade Kubernetes — change `kubernetesVersion` in the `control-plane-manager` ModuleConfig instead.

  If `Automatic` is specified, then the control plane version is used, which is considered stable at the moment. If the stable version of control plane is less than the maximum version that has ever been installed in the cluster, more than 1 minor version, then the version of the cluster will not be changed. The version may change when the minor version of the Deckhouse release is changed (see a corresponding release message).

  Allowed values: `1.33`, `1.34`, `1.35`, `1.36`, `1.37`, `Automatic`

- **podSubnetCIDR**
  Deprecated
  string

  Deprecated. Use `podSubnetCIDR` in the `network` section of the `control-plane-manager` ModuleConfig instead.

  Address space of the cluster’s Pods.

  Must be an address with a prefix length, for example `10.111.0.0/16`. A value written without one is rejected when the configuration is loaded.

> **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **podSubnetNodeCIDRPrefix**
  Deprecated
  string

  Deprecated. Use `podSubnetNodeCIDRPrefix` in the `network` section of the `control-plane-manager` ModuleConfig instead.

  The prefix size of the Pod network allocated to each node.

  This parameter determines how many IP addresses are available for Pods on each individual node. For example, if `podSubnetCIDR: 10.111.0.0/16` and `podSubnetNodeCIDRPrefix: "24"`, each node will receive a `/24` subnet (e.g., `10.111.0.0/24`, `10.111.1.0/24`, etc.), providing up to 254 IP addresses per node (253 usable for Pods, accounting for network and broadcast addresses).

  Calculation examples:

  - With `/16` network and `/24` prefix: maximum **256 nodes** (2^(24-16) = 256)
  - With `/16` network and `/25` prefix: maximum **512 nodes**, but only **126 IPs per node**
  - Each Pod network IP can support up to 2 Pods (one running + one terminating)

  > **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

  If the parameter is set in neither place, `24` is used.

- **proxy**
  object

  Available in editions: BE, SE, SE+, Ultimate/EE

  Global proxy setup (mainly for working in air-gapped environments).

  The parameters described in this section will be translated into the environment variables `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` for all cluster nodes and Deckhouse components. This will result in HTTP(S) requests (curl, git, registry, etc.) to all resources not listed in the `noProxy` parameter being made through a proxy. Note that the `podSubnetCIDR` and `serviceSubnetCIDR` subnets, as well as the `clusterDomain` domain are added to `noProxy` automatically.

  > **Caution!** To avoid using proxies in requests between pods and services located in the cluster node network, make sure you list all the host subnets in the `noProxy` parameter.

  - **proxy.httpProxy**
    string

    Available in editions: BE, SE, SE+, Ultimate/EE

    Proxy URL for HTTP requests.

    If necessary, specify the proxy server’s username, password, and port.

    Pattern: `^https?://([!*'();&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+(\:[!*'();:@&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+)?@)?[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?$`

    Examples:

    ```
    httpProxy: http://proxy.company.my
    ```

    ```
    httpProxy: https://user:password@proxy.company.my:8443
    ```

    ```
    httpProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
    ```

    ```
    httpProxy: https://user%40domain.local:password@proxy.company.my:8443
    ```

  - **proxy.httpsProxy**
    string

    Available in editions: BE, SE, SE+, Ultimate/EE

Proxy URL for HTTPS requests.

If necessary, specify the proxy server's username, password, and port.

Pattern: `^https?://([!*'();&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+(\:[!*'();:@&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+)?@)?[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?$`

Examples:

```
httpsProxy: http://proxy.company.my
```

```
httpsProxy: https://user:password@proxy.company.my:8443
```

```
httpsProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
```

```
httpsProxy: https://user%40domain.local:password@proxy.company.my:8443
```

- **proxy.noProxy**
  array of strings

  Available in editions: BE, SE, SE+, Ultimate/EE

  List of no proxy IP and domain entries.

  For wildcard domains, use a domain name with a dot prefix, e.g., ".example.com".

  > **Caution.** If the cluster is supposed to have pods interacting with services located in the cluster node network, then specify the list of subnets that are used on the nodes.

  - **Element of the array**
    string

    Pattern: `^[a-z0-9\-\./]+$`

- **serviceSubnetCIDR**
  Deprecated
  string

  Deprecated. Use `serviceSubnetCIDR` in the `network` section of the `control-plane-manager` ModuleConfig instead.

  Address space of the cluster's services.

  Must be an address with a prefix length, for example `10.222.0.0/16`. A value written without one is rejected when the configuration is loaded.

  > **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

CNIMigration

Short names: cnim

Scope: Cluster
Version: v1alpha1

Defines the configuration for CNI migration.

- **spec**
  object

  Defines the desired state of CNI migration.

  - **spec.targetCNI**
    string

    The target CNI to migrate to.

- **status**
  object

  Defines the observed state of CNI migration.

  - **status.conditions**
    array of objects

Reflect the state of the migration as a whole.

- **status.conditions.lastTransitionTime**
  string

  Required value
- **status.conditions.message**
  string

  Required value
- **status.conditions.reason**
  string

  Required value
- **status.conditions.status**
  string

  Required value
- **status.conditions.type**
  string

  Required value

- **status.currentCNI**
  string

  The detected CNI from which the switch is being made.
- **status.failedSummary**
  array of objects

  Details about nodes that failed the migration.

  - **status.failedSummary.node**
    string
  - **status.failedSummary.reason**
    string
- **status.nodesFailed**
  integer

  The number of nodes where an error occurred.
- **status.nodesSucceeded**
  integer

  The number of nodes that have successfully completed the current phase.
- **status.nodesTotal**
  integer

  The total number of nodes involved in the migration.
- **status.phase**
  string

  The current step of the migration process.

CNINodeMigration

Short names: cninm

Scope: Cluster
Version: v1alpha1

Defines the configuration for CNI migration on a specific node.

- **spec**
  object

  Defines the desired state of CNI migration on a node.
- **status**
  object

  Defines the observed state of CNI migration on a node.

  - **status.conditions**
    array of objects

    A list of conditions for this migration.

    - **status.conditions.lastTransitionTime**
      string

      Required value
    - **status.conditions.message**

string

Required value

- **status.conditions.reason**
  string

  Required value

- **status.conditions.status**
  string

  Required value

- **status.conditions.type**
  string

  Required value

- **status.phase**
  string

  The current phase of the migration on the node.

DeckhouseRelease

Scope: Cluster
Version: v1alpha1

Determines the state and parameters for applying a specific release (version) of the Deckhouse Platform (DP) in the cluster.

DeckhouseRelease objects are automatically created by DP upon the discovery of new DP versions on the selected release channel. Modifying DeckhouseRelease enables the management of the process for applying the corresponding DP version. More details about configuring DP updates can be found in the documentation.

The current versions of DP and modules by release channels can be found at releases.deckhouse.ru.

- **approved**
  boolean

  Allows or disables manual updates.

  Used only if the `deckhouse` module is configured for manual update mode (update.mode parameter is set to `Manual`). Ignored if the update mode is set to `Auto` or `AutoPatch`.

  For more information on confirming manual updates, refer to the documentation.

  Default: `false`

- **spec**
  object

  Required value

  - **spec.applyAfter**
    string

    Marks release as a part of canary-release. This release will be delayed until the specified time. If the release is waiting to be applied, check `.status.message` for the reason.

  - **spec.changelog**
    object

    A structure containing a list of DP changes (and modules included in the DP) of the release.

    For more information about DP release changelogs, refer to Updating.

  - **spec.changelogLink**
    string

    Link to site with full changelog for this release.

  - **spec.disruptions**
    Deprecated
    array of strings

    Disruptive changes in the release.

  - **spec.requirements**
    object

    A structure containing a list of requirements for installing the release. It is used by the DP core. If the requirements are not met, the release may be skipped or blocked from installation.

    Reports on unmet requirements can be found in the field `.status.message` of the DeckhouseRelease object.

  - **spec.version**
    string

    Required value

DP version.

Example:

```
version: v1.73.2
```

- **status**
  object
  - **status.approved**
    boolean

    The status of the release’s readiness for deployment. It makes sense only for Manual updates (`update.mode: Manual`).
  - **status.message**
    string

    Detailed status or error message.
  - **status.phase**
    string

    Current status of the release.

    Allowed values: `Pending`, `Deployed`, `Outdated`, `Suspended`, `Superseded`, `Skipped`
  - **status.transitionTime**
    string

    Time of release status change.

InitConfiguration

Version: deckhouse.io/v1

Deckhouse configuration to start after installation.

Example:

```
apiVersion: deckhouse.io/v1
kind: InitConfiguration
deckhouse:
  imagesRepo: nexus.company.my/deckhouse/ee
  registryDockerCfg:
eyJhdXRocyI6IHsgIm5leHVzLmNvbXBhbnkubXkiOiB7InVzZXJuYW1lIjoibmV4dXMtdXNlciIsInBhc3N3b3JkIjoibmV4dX
MtcEBzc3cwcmQiLCJhdXRoIjoiYm1WNGRYTXRkWE5sY2pwdVpYaDFjeTF3UUhOemR6QnlaQW89In19fQo=
  registryScheme: HTTPS
  registryCA: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`
- **deckhouse**
  Deprecated
  object

  Required value

  Initial parameters required to install Deckhouse.
  - **deckhouse.devBranch**
    Deprecated
    string

    The parameter is used for development needs. Will be replaced with the CLI-tools.
  - **deckhouse.imagesRepo**
    string

    Address of a container registry with Deckhouse images.

    Specify it if Deckhouse Enterprise Edition edition or third-party registry (e.g. proxy server in a closed environment) is used.

    The address matches the edition of Deckhouse used. The public container registry address for Deckhouse Enterprise Edition is `registry.deckhouse.io/deckhouse/ee`.

Default: `registry.deckhouse.io/deckhouse/ce`

Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_\/]+)?$`

- **deckhouse.registryCA**
  string

  Root CA certificate to validate the container registry's HTTPS certificate (if self-signed certificates are used).

- **deckhouse.registryDockerCfg**
  string

  A Base64-encoded string from the Docker client configuration file (in Linux it is usually `$HOME/.docker/config.json`), for accessing a third-party container registry.

  For example, to access the container registry `registry.company.my` under the user `username` with the password `password` it will be `eyJhdXRocyI6eyJyZWdpc3RyeS5jb21wYW55Lm15Ijp7ImF1dGgiOiJkWE5sY201aGJXVTZjR0Z6YzNkdmNtUT0ifX19Cg==` (string `{"auths":{"registry.company.my":{"auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}` in Base64).

  Default:

  Example:

  ```
  registryDockerCfg: eyJhdXRocyI6IHsgInJlZ2lzdHJ5LmRlY2tob3VzZS5pbyI6IHt9fX0=
  ```

- **deckhouse.registryScheme**
  string

  Registry access scheme (HTTP or HTTPS).

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **kind**
  string

  Required value

  Allowed values: `InitConfiguration`

Module

Scope: Cluster

- v1beta1
- v1alpha1

Module represents a module instance managed via the package system.

- **spec**
  object

  Required value

  Spec defines the behavior of a Module.

  - **spec.enabled**
    boolean

    Enables or disables the module. Unset leaves the decision to the platform.

  - **spec.maintenance**
    string

    Defines the module maintenance mode.

    - `NoResourceReconciliation`: A mode for developing or tweaking the module.

      In this mode:

      - Configuration or hook changes are not reconciled, which prevents resources from being updated automatically.
      - Resource monitoring is disabled, which prevents deleted resources from being restored.
      - All the module's resources are labeled with `maintenance.deckhouse.io/no-resource-reconciliation`.
      - The `ModuleIsInMaintenanceMode` alert is triggered.

    Allowed values: `NoResourceReconciliation`

  - **spec.packageRepositoryName**
    string

    Name of the repository where the package is located. If not specified, the default repository is used.

    Example:

    ```
    packageRepositoryName: deckhouse
    ```

- **spec.packageVersion**
  string

  Required value

  Version of the module package to install.

  Example:

  ```
  packageVersion: v1.0.0
  ```

- **spec.releaseChannel**
  string

  Release channel for the module package.

- **spec.settings**
  object

  Configuration settings for the module.

- **spec.settingsVersion**
  number

  Version of the settings schema. Distinct from packageVersion, which selects the module package.

- **spec.update**
  object

  Module update settings: update mode, update windows, and blocking updates when alerts are present.

  If this parameter is not specified, the Deckhouse Platform update settings are applied.

  Example:

  ```
  update:
    mode: Auto
    windows:
    - from: '8:00'
      to: '15:00'
      days:
      - Tue
      - Sat
    blockOnAlerts:
      enabled: true
      severity: 2
  ```

  - **spec.update.blockOnAlerts**
    object

    Settings for blocking module updates while there are alerts in the cluster.

    - **spec.update.blockOnAlerts.enabled**
      boolean

      Indicates whether to block module updates when the cluster has alerts with a severity level equal to or higher than the value specified in the `severity` parameter.

    - **spec.update.blockOnAlerts.severity**
      integer

      Alert severity level at which updates are blocked.

      Set as a value from `0` to `9`, where `0` is the highest severity and `9` is the lowest.

      Default: `4`

      Allowed values: `0 <= X <= 9`

  - **spec.update.mode**
    string

    Update mode of the module on its release channel.

    - `AutoPatch`: Automatic update mode for patch versions.

      To change a minor version (for example, from `v1.2.*` to `v1.3.*`), confirmation is required.

      A patch version update (for example, from `v1.3.1` to `v1.3.2`) is applied taking into account the update windows, if they are set.

    - `Auto`: Automatic update mode for all versions.

      Minor version updates (for example, from `v1.2.*` to `v1.3.*`) and patch version updates (for example, from `v1.3.1` to `v1.3.2`) are applied taking into account the update windows, if they are set.

- `Manual`: Manual update mode for all versions.

  Confirmation is required for updating both minor and patch versions.

Default: `AutoPatch`

Allowed values: `AutoPatch`, `Auto`, `Manual`

Examples:

```
mode: AutoPatch
```

```
mode: Auto
```

```
mode: Manual
```

- **spec.update.windows**
  array of objects

  List of module update windows.

  Time interval during the day when module updates are allowed.

  - **spec.update.windows.days**
    array of strings

    Days of the week on which the update window is applied.

    If the parameter is omitted, the update window is applied every day.

    - **Element of the array**
      string

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

  - **spec.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be earlier than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

    Should be later than the start time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

- **status**
  object

  Status of a Module.

  - **status.conditions**
    array of objects

    Conditions reflecting the latest observations of the module state.

    Condition contains details for one aspect of the current state of this API Resource.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**

string

Required value

message is a human readable message indicating details about the transition. This may be an empty string.

Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.currentVersion**
  object

  Information about the currently installed version.

  - **status.currentVersion.channel**
    string

    Release channel from which the version was installed.

  - **status.currentVersion.version**
    string

    Semantic version of the installed module.

- **status.enabled**
  boolean

  Reports whether the module is enabled: `spec.enabled` resolved against the edition bundle, the enabled-script and the module dependencies. It can differ from `spec.enabled` — an explicitly enabled module stays disabled while a module it requires is disabled. Absent until the first decision.

- **status.lastAppliedConfiguration**
  object

  LastAppliedConfiguration is the effective settings (user configuration merged with config-schema defaults) that drove the most recent successful apply.

- **status.summary**
  object

  Summary aggregates the high-level user-facing state, message and resolution hint for the module. The controller always populates it on reconcile — every module maps to exactly one lifecycle state — so it is the single source of truth for the UI; clients should not re-derive these values from the conditions. The pointer leaves it absent only before the first status computation.

  - **status.summary.message**
    string

Message is a human-readable description of the current state.

- **status.summary.state**
  string

  State is the high-level lifecycle state observed for the module. Always one of: Pending, Failed, Updating, Configuring, Ready, Degraded, Suspended, Deleting.

  Examples:

  ```
  state: Pending
  ```

  ```
  state: Failed
  ```

  ```
  state: Updating
  ```

  ```
  state: Configuring
  ```

  ```
  state: Ready
  ```

  ```
  state: Degraded
  ```

  ```
  state: Suspended
  ```

  ```
  state: Deleting
  ```

- **status.summary.tip**
  string

  Tip is a human-readable instruction on how to resolve the current state. Empty when no action is required.

- **status.tracking**
  object

  Nelm tracking.

Describes the module's status in the cluster. The `Module` object is created automatically after configuring the ModuleSource and successfully completing synchronization.

- **properties**
  object
  - **properties.accessibility**
    object

    Module accessibility settings.

    - **properties.accessibility.editions**
      object

      Module operation settings in Deckhouse editions.

      - **properties.accessibility.editions.<KEY_NAME>**
        object

        `<KEY_NAME>` — item (key) name.

        - **properties.accessibility.editions.<KEY_NAME>.available**
          boolean
        - **properties.accessibility.editions.<KEY_NAME>.enabledInBundles**
          array of strings
  - **properties.availableSources**
    array of strings

    Available sources for downloading the module.

  - **properties.critical**
    boolean

    Indicates whether the module critical or not.

  - **properties.disableOptions**
    object

    Parameters of module disable protection.

    - **properties.disableOptions.confirmation**
      boolean
    - **properties.disableOptions.message**
      string

      Deprecated: use Messages with localized ru/en fields instead.

        - **properties.disableOptions.messages**
          object

          Warning messages displayed when disabling the module, localized for different languages. If a message for the selected language is not defined, the value of the `message` field is used instead.

          - **properties.disableOptions.messages.en**
            string
          - **properties.disableOptions.messages.ru**
            string
      - **properties.exclusiveGroup**
        string

        Indicates the group where only one module can be active at a time.

      - **properties.namespace**
        string

        Module namespace.

      - **properties.releaseChannel**
        string

        Module release channel.

      - **properties.requirements**
        object

        Module dependencies, a set of requirements that must be met for Deckhouse Kubernetes Platform (DKP) to run the module.

        - **properties.requirements.bootstrapped**
          string

          Required cluster installation status (for built-in DKP modules only).

        - **properties.requirements.deckhouse**
          string

          Required Deckhouse version.

        - **properties.requirements.kubernetes**
          string

          Required Kubernetes version.

        - **properties.requirements.modules**
          object

          A list of other enabled modules required for the module.

      - **properties.source**
        string

        Source the module was downloaded from (otherwise will be blank).

      - **properties.stage**
        string

        Current stage of the module lifecycle.

      - **properties.subsystems**
        array of strings

        Module subsystems.

      - **properties.updatePolicy**
        string

        Module update policy.

      - **properties.version**
        string

        Module version.

      - **properties.weight**
        integer

        Module *weight* (priority).

- **spec**
  object
- **status**
  object
    - **status.conditions**
      array of objects
      - **status.conditions.lastProbeTime**
        string
      - **status.conditions.lastTransitionTime**
        string

        - **status.conditions.message**
          string
        - **status.conditions.reason**
          string
        - **status.conditions.status**
          string
        - **status.conditions.type**
          string
    - **status.hooksState**
      string

      Hooks status report.

    - **status.phase**
      string

      Module phase.

      Allowed values: `Unavailable`, `Available`, `Downloading`, `DownloadingError`, `Reconciling`, `Installing`, `HooksDisabled`, `WaitSyncTasks`, `Downloaded`, `Conflict`, `Ready`, `Error`

      Examples:

      ```
      phase: Unavailable
      ```

      ```
      phase: Available
      ```

      ```
      phase: Downloading
      ```

      ```
      phase: DownloadingError
      ```

      ```
      phase: Reconciling
      ```

      ```
      phase: Installing
      ```

      ```
      phase: HooksDisabled
      ```

      ```
      phase: WaitSyncTasks
      ```

      ```
      phase: Downloaded
      ```

      ```
      phase: Conflict
      ```

      ```
      phase: Ready
      ```

      ```
      phase: Error
      ```

ModuleConfig

Short names: mc

Scope: Cluster
Version: v1alpha1

Defines the configuration of the Deckhouse Platform module (module parameters). The name of the ModuleConfig resource must match the name of the module (for example, `control-plane-manager` for the `control-plane-manager` module).

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-1
spec:
  enabled: true
  settings: {}
  version: 1
```

- **spec**
  object

  Required value

- **spec.enabled**
  boolean

  Enables or disables the module.

  Example:

  ```
  enabled: false
  ```

- **spec.maintenance**
  string

  Defines the module maintenance mode.

  - `NoResourceReconciliation`: A mode for developing or tweaking the module.

    In this mode:

    - Configuration or hook changes are not reconciled, which prevents resources from being updated automatically.
    - Resource monitoring is disabled, which prevents deleted resources from being restored.
    - All the module's resources are labeled with `maintenance: NoResourceReconciliation`.
    - The `ModuleIsInMaintenanceMode` alert is triggered.

  Allowed values: `NoResourceReconciliation`

  Example:

  ```
  maintenance: NoResourceReconciliation
  ```

- **spec.settings**
  object

  Module settings.

- **spec.source**
  string

  The source of the module it provided by one (otherwise empty).

- **spec.updatePolicy**
  string

  Module update policy.

  Example:

  ```
  updatePolicy: test-alpha
  ```

- **spec.version**
  number

  Version of settings schema.

  Example:

  ```
  version: 1
  ```

- **status**
  object
  - **status.message**
    string

    Additional information

  - **status.version**
    string

    Version of settings schema in use

ModulePullOverride

Short names: mpo

Scope: Cluster

- v1alpha2
- v1alpha1

Defines the resource configuration for downloading specific versions of Deckhouse modules.

> **i** **Caution**. This resource is intended for development and debugging environments only. Using it in production clusters is not recommended. Support for the resource might be removed in future Deckhouse Platform versions.

- **spec**
  object

  Required value

  - **spec.imageTag**
    string

    Required value

    Module container image tag, which will be pulled.

  - **spec.rollback**
    boolean

    Indicates whether the module release should be rollback after deleting mpo.

    Default: `false`

  - **spec.scanInterval**
    string

    Scan interval for checking the image digest. If the digest changes, the module is updated. Supported units: `h`, `m`, `s`.

    Default: `15s`

    Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

- **status**
  object
  - **status.imageDigest**
    string

    Digest of the module image.

  - **status.message**
    string

    Details of the resource status.

  - **status.updatedAt**
    string

    When the module was last updated.

  - **status.weight**
    integer

    Module weight.

Deprecated resource. Support for the resource might be removed in a later release.

Defines the configuration.

- **spec**
  object

  Required value

  - **spec.imageTag**
    string

    Required value

    Module container image tag, which will be pulled.

  - **spec.rollback**
    boolean

    Indicates whether the module release should be rollback after deleting ModulePullOverride.

    Default: `false`

  - **spec.scanInterval**
    string

    Scan interval for checking the image digest. If the digest changes, the module is updated. Supported units: `h`, `m`, `s`.

    Default: `15s`

    Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

  - **spec.source**
    string

    Required value

    Reference to the ModuleSource with the module.

- **status**

object
- **status.imageDigest**
  string

  Digest of the module image.
- **status.message**
  string

  Details of the resource status.
- **status.updatedAt**
  string

  When the module was last updated.
- **status.weight**
  integer

  Module weight.

ModuleRelease

Short names: mr

Scope: Cluster
Version: v1alpha1

Defines the configuration for a Deckhouse release.

ModuleRelease resources are created by Deckhouse.

- **spec**
  object

  Required value
  - **spec.applyAfter**
    string

    Time until which the release will be delayed.
  - **spec.changelog**
    object

    Release’s changelog for the module.
  - **spec.moduleName**
    string

    Required value

    Module name.
  - **spec.requirements**
    object

    Release dependencies, a set of requirements that must be met for Deckhouse Platform to run the module release.
    - **spec.requirements.deckhouse**
      string

      Required Deckhouse version.
    - **spec.requirements.kubernetes**
      string

      Required Kubernetes version.
    - **spec.requirements.modules**
      object

      A list of other modules required for the module release. Ensure the modules are enabled.
  - **spec.update**
    object

    Optional transition rules.
    - **spec.update.versions**
      array of objects

      List of `from` → `to` transition rules that allow skipping step-by-step updates. If the current installed module version (status `Deployed`) is not lower than `from`, and the cluster has a release whose version matches `to`, the controller will skip intermediate releases and update the module to the version from `to`. The `to` value can specify a minor line (`X.Y` — the latest available `X.Y.Z` will be selected). The rule is specified in the constrained release — the one whose version matches `to`.
      - **spec.update.versions.from**
        string

Required value

The minimum version from which the transition is allowed (format `X.Y`).

- **spec.update.versions.to**
  string

  Required value

  The end version of the range — a minor line (`X.Y`).

- **spec.version**
  string

  Required value

  Module version.

  Example:

  ```
  version: v1.0.0
  ```

- **spec.weight**
  integer

  Module *weight* (priority).

- **status**
  object
  - **status.approved**
    boolean

    Status indicating that the release is ready for deployment. For the Manual update mode only (`update.mode: Manual`).

  - **status.message**
    string

    Detailed status or error message.

  - **status.phase**
    string

    Current status of the release.

    Allowed values: `Pending`, `Deployed`, `Superseded`, `Suspended`, `Skipped`, `Terminating`

  - **status.pullDuration**
    string

    Module loading duration.

  - **status.size**
    integer

    Size of the module image.

  - **status.transitionTime**
    string

    Time of release status change.

ModuleSettingsDefinition

Scope: Cluster
Version: v1alpha1

It displays module settings. Defines a list of module settings versions.

- **spec**
  object

  Required value

  Specification of the module settings.

  - **spec.versions**
    array of objects

    List of module settings versions. Each version includes a name and a schema.

    - **spec.versions.conversions**
      array of objects

      List of conversion rules for this version.

      A single conversion rule with expressions and descriptions.

      - **spec.versions.conversions.descriptions**
        object

Localized descriptions of the conversion.

- **spec.versions.conversions.descriptions.en**
  string

  English description of the conversion.

- **spec.versions.conversions.descriptions.ru**
  string

  Russian description of the conversion.

- **spec.versions.conversions.expr**
  array of strings

  Array of jq expressions to transform settings.

- **spec.versions.name**
  string

  Required value

  Module settings version.

- **spec.versions.schema**
  object

  Settings schema for the given module version.

ModuleSource

Short names: ms

Scope: Cluster
Version: v1alpha1

Defines the configuration of a source of Deckhouse modules.

For more information about installing the module from the source, see the section “Running the module in the DP cluster”.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.io/modules-source
    dockerCfg: "<base64 encoded credentials>"
```

- **spec**
  object

  Required value

  - **spec.registry**
    object

    Required value

    - **spec.registry.ca**
      string

      Root CA certificate (PEM format) to validate the registry’s HTTPS certificate (if self-signed certificates are used).

      > **i** Creating a ModuleSource resource with the CA certificate spec will cause the container to restart on all nodes.

    - **spec.registry.dockerCfg**
      string

      Container registry access token in Base64. If using anonymous access to the container registry, do not fill in this field.

    - **spec.registry.repo**
      string

      Required value

      URL of the container registry.

      Example:

```
repo: registry.example.io/deckhouse/modules
```

- **spec.registry.scheme**
  string

  Protocol to access the registry.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.releaseChannel**
  Deprecated
  string

  Desirable default release channel for modules in the current source.

- **spec.scanInterval**
  string

  Interval for registry scan.

  Defines the frequency of checking the container registry for new modules and their versions.

  Defaults to 30 minutes. Explicit intervals shorter than 3 minutes are rounded up to 3 minutes.

  Default: `30m`

  Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

  Examples:

```
scanInterval: 5m
```

```
scanInterval: 1h
```

```
scanInterval: 6h30m
```

- **status**
  object
  - **status.message**
    string
  - **status.modules**
    array of objects

    The list of modules available from the source and their update policies.

    - **status.modules.checksum**
      string

      The module checksum.

    - **status.modules.error**
      string

      The module processing error.

    - **status.modules.name**
      string

      The module name.

    - **status.modules.overridden**
      boolean

      If ModulePullOverride for this module exists.

    - **status.modules.policy**
      string

      The module policy name.

    - **status.modules.pullError**
      string

      The module pull error.

    - **status.modules.version**
      string

      The module version.

  - **status.modulesCount**
    integer

    The number of modules available.

  - **status.phase**

string

The current phase.

Allowed values: `Active`, `Terminating`

- **status.syncTime**
  string

  When the repository was synchronized.

ModuleUpdatePolicy

Short names: mup

Scope: Cluster

- v1alpha2
- v1alpha1

Defines the update settings for a module's release.

Example:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - Mon
      - Wed
      from: '13:30'
      to: '14:00'
```

- **spec**
  object

  Required value

  - **spec.releaseChannel**
    string

    Desirable module release channel.

    The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

    Default: `Stable`

    Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `Lts`

  - **spec.update**
    object

    Required value

    Update settings for target modules.

    - **spec.update.mode**
      string

      Modules version update mode (release change).

      - `AutoPatch` — automatic update mode for patch releases.

        To change a minor version (for example, from `v1.15.*` to `v1.16.*`), confirmation is required.

        A patch version update (for example, from `v1.16.1` to `v1.16.2`) is applied according to the update windows, if they are set.

      - `Auto` — all updates are applied automatically.

        Modules minor version updates (for example, from `v1.15.*` to `v1.16.*`) and patch version updates (for example, from `v1.16.1` to `v1.16.2`) are applied according to the update windows or (if no update windows are set) as they appear on the corresponding release channel;

      - `Manual` — confirmation is required for updating both minor and patch versions.

      To confirm the update, add the `modules.deckhouse.io/approved="true"` annotation to the corresponding ModuleRelease resource.

      Default: `Auto`

Allowed values: `Auto`, `Manual`, `AutoPatch`

- **spec.update.windows**
  array of objects

  Modules update timetable.

  - **spec.update.windows.days**
    array of strings

    The days of the week on which the update window is applied.

    Examples:

    ```
    days: Mon
    ```

    ```
    days: Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

      Example:

      ```
      Mon
      ```

  - **spec.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be less than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

    Should be more than the start time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

Deprecated resource. Support for the resource might be removed in a later release.

Defines the update settings for a module’s release.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  moduleReleaseSelector:
    labelSelector:
      matchLabels:
        source: example
        module: module-1
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - Mon
      - Wed
      from: '13:30'
      to: '14:00'
```

- **spec**
  object

  Required value

  - **spec.moduleReleaseSelector**
    object

    Required value

    Selects target modules to apply update settings to.

    - **spec.moduleReleaseSelector.labelSelector**
      object

      Required value

      Label-selector-based filter to match target modules.

      If both `matchExpressions` and `matchLabels` parameters are set, their requirements are ANDed together — they must all be satisfied in order to match. If multiple `matchExpression` conditions are provided, they all must be satisfied in order to match.

      - **spec.moduleReleaseSelector.labelSelector.matchExpressions**
        array of objects

        An array of set-based expressions.

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.key**
          string

          Required value

          A label name.

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.operator**
          string

          Required value

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.values**
          array of strings

          A label value.

      - **spec.moduleReleaseSelector.labelSelector.matchLabels**
        object

        A number of equality-based label filters.

        Example:

        ```
        matchLabels:
          source: deckhouse
          module: deckhouse-admin
        ```

  - **spec.releaseChannel**
    string

    Desirable module release channel.

The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

Default: `Stable`

Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`

- **spec.update**
  object

  Required value

  Update settings for target modules.

  - **spec.update.mode**
    string

    Modules version update mode (release change).

    - `AutoPatch` — automatic update mode for patch releases.

      To change a minor version (for example, from `v1.15.*` to `v1.16.*`), confirmation is required.

      A patch version update (for example, from `v1.16.1` to `v1.16.2`) is applied according to the update windows, if they are set.

    - `Auto` — all updates are applied automatically.

      Modules minor version updates (for example, from `v1.15.*` to `v1.16.*`) and patch version updates (for example, from `v1.16.1` to `v1.16.2`) are applied according to the update windows or (if no update windows are set) as they appear on the corresponding release channel;

    - `Manual` — confirmation is required for updating both minor and patch versions.

    To confirm the update, add the `modules.deckhouse.io/approved="true"` annotation to the corresponding ModuleRelease resource.

    - `Ignore` — updates are ignored.

    Default: `AutoPatch`

    Allowed values: `Auto`, `Manual`, `Ignore`, `AutoPatch`

  - **spec.update.windows**
    array of objects

    Modules update timetable.

    - **spec.update.windows.days**
      array of strings

      The days of the week on which the update window is applied.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

        Example:

        ```
        Mon
        ```

    - **spec.update.windows.from**
      string

      Required value

      Start time of the update window (UTC timezone).

      Should be less than the end time of the update window.

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      from: '13:00'
      ```

    - **spec.update.windows.to**
      string

Required value

End time of the update window (UTC timezone).

Should be more than the start time of the update window.

Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Example:

```
to: '18:30'
```

PackageRepository

Scope: Cluster
Version: v1alpha1

PackageRepository defines a package repository available in Deckhouse Platform.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepository
metadata:
  name: example
spec:
  registry:
    ca: "<PEM-encoded CA certificate>"
    dockerCfg: "<base64 encoded credentials>"
    login: admin
    password: "<password>"
    repo: registry.example.io/packages
    scheme: https
  scanInterval: 5m
```

- **spec**
  object

  Required value

  Package repository configuration.

  - **spec.registry**
    object

    Required value

    Access parameters for the container registry where the package repository is located.

    - **spec.registry.ca**
      string

      Root certificate in PEM format used to verify the TLS certificate of the container registry when connecting over HTTPS.

      Used if the registry uses self-signed SSL certificates.

      Example:

      ```
      ca: "<PEM-encoded CA certificate>"
      ```

    - **spec.registry.dockerCfg**
      string

      Container registry access configuration in `~/.docker/config.json` format, encoded in Base64.

      Leave this field empty for anonymous access.

      Example:

      ```
      dockerCfg: "<base64 encoded credentials>"
      ```

    - **spec.registry.login**
      string

      Username for authenticating to the container registry.

      Example:

      ```
      login: admin
      ```

    - **spec.registry.password**

string

Password for authenticating to the container registry.

Example:

```
password: "<password>"
```

- **spec.registry.repo**
  string

  Required value

  Address of the package repository in the container registry.

  Example:

  ```
  repo: registry.example.io/packages
  ```

- **spec.registry.scheme**
  string

  Protocol used to access the package repository, for example, `https`.

  Example:

  ```
  scheme: https
  ```

- **spec.scanInterval**
  string

  Interval between automatic scans of the package repository for discovering new packages and versions.

  Default: `6h`

  Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

  Examples:

  ```
  scanInterval: 5m
  ```

  ```
  scanInterval: 1h
  ```

  ```
  scanInterval: 6h30m
  ```

- **status**
  object

  Current state of the package repository.

  - **status.conditions**
    array of objects

    Conditions reflecting the current state of the package repository.

    Condition reflecting an aspect of the current state of the package repository.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition status last changed.

    - **status.conditions.message**
      string

      Required value

      Message with additional information about the condition.

      May be empty.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Value of `.metadata.generation` for which the condition was set.

      Indicates whether the condition corresponds to the current generation of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

Required value

Programmatic identifier indicating the reason for the last condition status change.

Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type in CamelCase or in the format `foo.example.com/CamelCase`.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.lastChangeTime**
  string

  Time of the last scan that discovered at least one new package version.

  If no new versions are discovered, the value is not updated.

- **status.lastNewVersions**
  integer

  Number of new package versions discovered during the last scan.

  Equals `0` if no new versions were discovered.

- **status.lastScanTime**
  string

  Time of the last successfully completed scan.

  If a scan fails, the value is not updated, and the error is reflected in the `LastScanSucceeded` condition.

- **status.message**
  string

  Message with additional information about the state of the package repository.

- **status.packages**
  array of objects

  List of packages available in the repository.

  - **status.packages.name**
    string

    Required value

    Package name.

  - **status.packages.type**
    string

    Required value

    Package type.

- **status.packagesCount**
  integer

  Total number of packages in the repository.

- **status.phase**
  string

  Current phase of the repository.

PackageRepositoryOperation

Short names: pro

Scope: Cluster
Version: v1alpha1

PackageRepositoryOperation represents an operation for scanning or updating a package repository.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepositoryOperation
metadata:
  name: example
spec:
  packageRepositoryName: deckhouse
  type: Update
  update:
    fullScan: false
    timeout: 5m
```

- **spec**
  object

  Required value

  Package repository operation parameters.

  - **spec.packageRepositoryName**
    string

    Required value

    Name of the package repository for which the operation is performed.

    Example:

    ```
    packageRepositoryName: deckhouse
    ```

  - **spec.type**
    string

    Required value

    Operation type.

    Allowed values: `Update`

  - **spec.update**
    object

    Update operation parameters.

    - **spec.update.fullScan**
      boolean

      Indicates whether to perform a full scan of the package repository.

    - **spec.update.timeout**
      string

      Operation timeout.

      Example:

      ```
      timeout: 5m
      ```

- **status**
  object

  Current state of the package repository operation.

  - **status.completionTime**
    string

    Time when the operation completed.

  - **status.conditions**
    array of objects

    Conditions reflecting the current state of the operation.

    The operation phase is determined by the `Completed` condition: while its status is `False`, the operation is in one of the intermediate phases (`Discover` or `Processing`). A `True` status indicates that the operation has completed with the `ScanSucceeded` or `ScanFailed` reason.

    Condition reflecting an aspect of the current state of the operation.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition status last changed.

- **status.conditions.message**
  string

  Required value

  Message with additional information about the condition.

  Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  Value of `.metadata.generation` for which the condition was set.

  Indicates whether the condition corresponds to the current generation of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  Programmatic identifier indicating the reason for the last condition status change.

  For the `Completed` condition, possible values are `Discover`, `Processing`, `ScanSucceeded`, and `ScanFailed`.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

  Examples:

  ```
  reason: Discover
  ```

  ```
  reason: Processing
  ```

  ```
  reason: Succeeded
  ```

  ```
  reason: Failed
  ```

- **status.conditions.status**
  string

  Required value

  Condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type in CamelCase or in the format `foo.example.com/CamelCase`.

  This resource uses the `Completed` condition, which indicates whether the operation has completed.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.packages**
  object

  Information about packages processed during the operation.

  - **status.packages.discovered**
    array of objects

    Packages discovered during the operation and awaiting processing.

    The list is empty after the operation completes.

    - **status.packages.discovered.name**
      string

      Required value

      Name of the discovered package.

  - **status.packages.failed**
    array of objects

    Packages that could not be processed.

- **status.packages.failed.errors**
  array of objects

  Required value

  Errors that occurred while processing the package.

  - **status.packages.failed.errors.message**
    string

    Required value

    Error message.

  - **status.packages.failed.errors.version**
    string

    Required value

    Package version for which the error occurred.

- **status.packages.failed.name**
  string

  Required value

  Name of the package that could not be processed.

- **status.packages.newVersionsOverall**
  integer

  Total number of new package versions discovered during the operation.

  A version is considered new if the corresponding ApplicationPackageVersion or ModulePackageVersion did not exist in the cluster.

  An ApplicationPackageVersion is also considered new if it was previously marked as missing from the container registry, but its image was discovered during the current operation.

- **status.packages.processed**
  array of objects

  Successfully processed packages.

  - **status.packages.processed.foundVersions**
    integer

    Number of package versions discovered during the operation.

  - **status.packages.processed.name**
    string

    Required value

    Name of the processed package.

  - **status.packages.processed.newVersions**
    integer

    Number of new package versions discovered during the operation.

    A version is considered new if the corresponding ApplicationPackageVersion or ModulePackageVersion did not exist in the cluster.

    An ApplicationPackageVersion is also considered new if it was previously marked as missing from the container registry, but its image was discovered during the current operation.

  - **status.packages.processed.type**
    string

    Package type.

- **status.packages.processedOverall**
  integer

  Total number of processed packages.

- **status.packages.total**
  integer

  Total number of discovered packages.

- **status.startTime**
  string

  Time when the operation started.

SSHConfig

Version: dhctl.deckhouse.io/v1

General dhctl SSH config.

Example:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHConfig
sshUser: user
sshPort: 22
sshExtraArgs: "-vvv"
sshAgentPrivateKeys:
- key: "<ssh-private-key>"
```

- **apiVersion**
  string

  Version of the Deckhouse API.

  Allowed values: `dhctl.deckhouse.io/v1`
- **kind**
  string

  Allowed values: `SSHConfig`
- **legacyMode**
  boolean

  Switch to legacy SSH mode (clissh).
- **modernMode**
  boolean

  Switch to modern SSH mode (gossh).
- **sshAgentPrivateKeys**
  array of objects
  - **sshAgentPrivateKeys.key**
    string

    Required value

    Contents of the private SSH key.

    This field does not support passing a certificate. To authenticate with a certificate in `gossh` mode, load the key together with its certificate into an ssh-agent accessible to dhctl through `SSH_AUTH_SOCK`. This method is unavailable when `legacyMode: true`.
  - **sshAgentPrivateKeys.passphrase**
    string

    Password for SSH key.
- **sshBastionHost**
  string

  SSH bastion host.
- **sshBastionPassword**
  string

  A password for the bastion user.
- **sshBastionPort**
  integer

  Port of SSH bastion.
- **sshBastionUser**
  string

  Username for bastion.
- **sshExtraArgs**
  string

  Additional arguments for SSH connection.
- **sshPort**
  integer

  SSH port.
- **sshUser**
  string

  SSH username.
- **sudoPassword**
  string

A sudo password for the user.

SSHHost

Version: dhctl.deckhouse.io/v1

General dhctl SSH host config.

Example:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHHost
host: 172.16.0.0
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `dhctl.deckhouse.io/v1`
- **host**
  string

  Required value

  Host.
- **kind**
  string

  Required value

  Allowed values: `SSHHost`

StaticClusterConfiguration

Version: deckhouse.io/v1

Parameters of a static (bare metal) cluster.

To change the `StaticClusterConfiguration` resource in a running cluster, run the following command:

```
d8 system edit static-cluster-configuration
```

Example:

```
apiVersion: deckhouse.io/v1
kind: StaticClusterConfiguration
internalNetworkCIDRs:
- 10.244.0.0/16
- 10.50.0.0/16
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`
- **internalNetworkCIDRs**
  array of strings

  List of internal cluster networks.

  Internal cluster networks connect Kubernetes components (`kube-apiserver`, `kubelet`, etc.).

  The parameter is mandatory in the following cases:

  - Cluster nodes have more than one network interface
  - The cluster is deployed inside a Deckhouse Virtualization Platform

  > **i** When changing the value, subnets in the new list must include IP addresses of currently joined nodes.

  Example:

```
internalNetworkCIDRs:
- 192.168.42.0/24
- 172.16.16.0/24
```

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **kind**
  string

  Required value

  Allowed values: `StaticClusterConfiguration`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Release channels

> For information on which versions of Deckhouse are available on which release channels as well as the planned date of the version update for a particular release channel, visit releases.deckhouse.io website.

Clusters, as infrastructure elements, usually have to meet various requirements.

A production cluster, unlike a development one, has higher requirements for reliability. In a production cluster, frequent component updates and changes are undesirable. All the cluster components must be thoroughly tested for stable and reliable operation.

Deckhouse uses **five release channels** which you can *soft-switch* between using the deckhouse module: just specify the desired release channel in the module configuration.

- Rock Solid
- Stable
- Early Access
- Beta
- Alpha

- The least stable release channel but with the most frequent appearance of new versions. Can bring new features but can be unstable.
- For more broad testing of new features to catch regressions
- **Recommended** release channel. Is mostly safe and can be used in non-critical environments or for local development
- Is mostly safe and we encourage you to use this version everywhere
- The most stable release channel and recommended for usage in critical environments with tight SLA

Additional resources

- releases.deckhouse.io
- Module `deckhouse` documentation

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Supported Kubernetes and OS versions

Linux

The following Linux distributions are currently supported for nodes:

Linux Distribution	Supported versions	Supported in redactions	Notes
		CE, BE, SE, SE+, EE	
CentOS	7	✓	

Linux Distribution	Supported versions	Supported in redactions	Notes
		CE, BE, SE, SE+, EE	
	8	✓	
	9	✓	
	10	✓	
Debian	**10**	✓	
	11, 12, 13	✓	
openSUSE	**15.4, 15.5, 15.6**	✓	
Rocky Linux	**8**	✓	
	9	✓	
Ubuntu	**18.04**	✓	
	20.04	✓	
	22.04, 24.04, 26.04	✓	

Kubernetes

The following Kubernetes versions are currently supported:

Version	Status
1.33	The version is not recommended for use. It is planned for removal in future releases.
1.34	Production-ready fully supported version.
1.35	**Default version.** Production-ready fully supported version.
1.36	Production-ready fully supported version.
1.37	Early access version, which is not suitable for production environments.

Container registry

i Deckhouse Platform works with any container registry supporting Basic or Bearer authentication and OCI image format specification.

Container registries that Deckhouse is guaranteed to work with
Sonatype Nexus Repository
Harbor
JFrog Artifactory
CNCF Distribution Registry (Docker Registry)
Project Quay*

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Kubernetes conformance test results

Deckhouse Platform is tested against the Kubernetes conformance suite of the CNCF Certified Kubernetes Conformance Program. The tests are run with Sonobuoy in `certified-conformance` mode for each Kubernetes minor version listed below.

> The conformance test results listed below are valid only when the `multitenancy-manager` module is disabled.

Test results

- Kubernetes **1.37** — Download JUnit XML report
- Kubernetes **1.36** — Download JUnit XML report
- Kubernetes **1.35** — Download JUnit XML report
- Kubernetes **1.34** — Download JUnit XML report
- Kubernetes **1.33** — Download JUnit XML report

CNCF Kubernetes conformance (Sonobuoy)

This directory stores **certified conformance** artifacts per Kubernetes minor version: `e2e.log` and `junit_01.xml` produced by Sonobuoy in `certified-conformance` mode.

Use it when you need to refresh bundled results after upgrading or validating the supported Kubernetes line.

---

1. Cluster and Deckhouse

Deploy a cluster with Deckhouse at the target Kubernetes version and ensure `kubectl` points at it.

---

2. Adjust RBAC and admission (required for a clean run)

Pod Security Standards can also block workloads that the e2e suite expects. For the run, relax **Admission Policy Engine** defaults so workloads are not constrained below what the suite needs.

Apply once (review before production clusters):

```
kubectl apply -f - <<'EOF'
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  enabled: true
  settings:
    podSecurityStandards:
      defaultPolicy: Privileged
  version: 1
EOF
```

---

3. Install Sonobuoy CLI

Download a release from Sonobuoy releases, unpack the binary, and put it on your `PATH` (or run from the unpack directory).

Example for Linux `amd64` (replace the version with the one you want):

```
curl -sL -o sonobuoy.tgz \
  'https://github.com/vmware-
tanzu/sonobuoy/releases/download/v0.57.3/sonobuoy_0.57.3_linux_amd64.tar.gz'
tar -xzf sonobuoy.tgz sonobuoy
chmod +x sonobuoy
```

---

4. Run conformance

```
./sonobuoy run --mode=certified-conformance --kubeconfig=/etc/kubernetes/super-admin.conf
```

Wait until `./sonobuoy status` reports the run as **completed**. To see progress, check that namespaces are being created and deleted (tests run in isolated namespaces).

The run typically takes **2 hours**.

---

5. Fetch only `e2e.log` and `junit_01.xml`

From the machine where the CLI runs:

```
./sonobuoy retrieve . -f sb.tar.gz
tar -xzf sb.tar.gz \
  plugins/e2e/results/global/e2e.log \
  plugins/e2e/results/global/junit_01.xml
```

You will get:

`plugins/e2e/results/global/e2e.log`
`plugins/e2e/results/global/junit_01.xml`

Optional: remove the tarball when done (`rm -f sb.tar.gz`).

Additional resources

- Download JUnit XML report

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse Platform editions comparison

The Deckhouse Platform is available in several editions. They are with different functions and support options.

Starting with version 1.76, the list of supported editions has been updated. Information about the supported editions and their features is being updated and will be available later.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Container registry structure

This page provides a general overview of the container image registries used to install the Deckhouse Platform (DP).

Commercial editions container registry

The registry contains modules and components available in the commercial DP editions:

- EE (address `registry.deckhouse.io/deckhouse/ee`).
- SE (address `registry.deckhouse.io/deckhouse/se`).
- SE+ (address `registry.deckhouse.io/deckhouse/se-plus`).
- BE (address `registry.deckhouse.io/deckhouse/be`).

Key features:

- implements functionality according to the edition;
- security testing is performed as part of the secure software development lifecycle;
- delivered electronically.

Images are provided with technical support, including operational consultation.

Image build and update specifics:

- built from open-source software included in DP:
  - from upstream (as-is);
  - with deep modernization (fork).
- includes proprietary company developments;
- vulnerability remediation and package updates are performed only when upstream open-source packages are updated;
- updates are provided:
  - with new functionality;

- with bug and vulnerability fixes;
- with security incident resolution.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Network interaction of the platform components

If the infrastructure where Deckhouse Platform (DP) is running has requirements to limit host-to-host network communications, the following conditions must be met:

- Tunneling mode for traffic between pods is enabled (configuration for CNI Cilium, configuration for CNI Flannel).
- Traffic between `podSubnetCIDR` encapsulated within a VXLAN is allowed (if inspection and filtering of traffic within a VXLAN tunnel is performed).
- If there is integration with external systems (e.g. LDAP, SMTP or other external APIs), it is required to allow network communication with them.
- Local network communication is fully allowed within each individual cluster node.
- Inter-node communication is allowed on the ports shown in the tables on the current page. Note that most ports are in the 4200-4299 range. When new platform components are added, they will be assigned ports from this range (if it is possible).

How to check the current VXLAN port...

```
d8 k -n d8-cni-cilium get cm cilium-config -o yaml | grep tunnel
```

Example output:

```
routing-mode: tunnel
tunnel-port: "4298"
tunnel-protocol: vxlan
```

> Changes related to the addition, removal, or reassignment of ports in the tables are listed in the “Network” section of a respective DP version on the Release notes page.

Requirements for network latency between master nodes

> DP supports HA mode using arbiter nodes. The network latency requirements listed below also apply to arbiter nodes.

Etcd only commits a writing after the change has been replicated to a majority of master nodes. Therefore, the round-trip time (RTT) between master nodes directly determines the cluster’s performance. A single operation in the cluster typically consists of several consecutive writes, so as latency increases, the application of manifests, operator operations, and Kubernetes API responsiveness slow down noticeably.

The sum of RTT and jitter between master nodes must not exceed 100 ms.

> ⚠ Round-trip time (RTT) is specified, not one-way delay: for example, the `ping` command returns RTT.
>
> Jitter—the variation in latency—is added to the RTT. For example, with an RTT of 90 ms and jitter of 20 ms, the total is 110 ms, and the requirement is not met, even though the RTT value itself is within the 100 ms limit.

When planning a cluster, it is recommended to aim for a latency of no more than 50 ms between future master nodes—that is, a twofold margin relative to the requirement. This margin is necessary for the following reasons:

- Pre-deployment measurements are performed on an idle network and may underestimate actual values.

- You need to account for latency spikes and variation, as well as possible infrastructure operations—for example, migrating a virtual machine to a zone with higher latency.

Where does the 100 ms value come from...

DP uses the standard etcd values:

- `ETCD_HEARTBEAT_INTERVAL` — 100 ms, the frequency at which the leader sends heartbeat messages to the other cluster members.
- `ETCD_ELECTION_TIMEOUT` — 1000 ms, the timeout for heartbeat messages; after this time, a cluster member begins leader re-election.

The etcd documentation requires that the election timeout be at least 10x RTT. Hence, the maximum allowed value is:

```
1000 ms / 10 = 100 ms
```

Potential issues

The main consequence of increased RTT is a proportional slowdown of the cluster. Network latency is directly reflected in write times: with an RTT of 300 ms, each write to etcd will take at least 300 ms. There is no threshold here; performance degrades continuously as latency increases.

The second issue is etcd leader re-elections. These are triggered not by the magnitude of the latency, but by its complete loss: a cluster member initiates an election if it does not receive messages from the leader within the time specified by `ETCD_ELECTION_TIMEOUT` (taking into account etcd's built-in randomization mechanism—from 1,000 to 2,000 ms). With a 100-ms heartbeat interval, this corresponds to 10–20 consecutive missed heartbeat messages. Typical causes:

- A network outage or packet loss lasting 1–2 seconds.
- Disk write delays on the leader node, preventing etcd from sending heartbeat messages in time.
- A restart of the etcd process.

Isolated spikes in latency do not trigger re-elections: heartbeat messages arrive, albeit late. Only writes that occur during the spike are delayed.

However, a high RTT makes re-elections protracted when they do occur. Votes are collected within a single RTT and must fit within `ETCD_ELECTION_TIMEOUT`. If the RTT is close to this timeout, the election does not complete in time and restarts, leaving the cluster leaderless. A tenfold margin is required to ensure the election succeeds on the first attempt.

During leader re-elections, etcd does not process write requests, so the Kubernetes API returns errors. The `KubeEtcdHighNumberOfLeaderChanges` alert signals frequent leader re-elections—more than three leader re-elections in 10 minutes.

Checking current latencies

Latency and its variation can be measured using `ping`. The command is run on one of the master nodes, or—when planning the cluster—on the server intended to become a master node. ICMP is sufficient for this check, as it is already included in the requirements for traffic between nodes:

```
ping -c 100 -i 0.2 <IP address of another master node>
```

Example of the resulting output:

```
rtt min/avg/max/mdev = 25.085/25.502/35.933/1.185 ms
```

Here, `avg` is the average RTT, `mdev` is the delay variation (jitter), and `max` is the actual maximum—that is, the RTT including spikes. There is no need to add the RTT and jitter manually: you should compare the `max` value against the requirements. When planning a cluster, we recommend staying within 50 ms; for a live cluster, do not exceed 100 ms.

Network latency at the other nodes

The control plane has no requirements regarding network latency on nodes that are not master nodes: latency on these nodes does not affect the cluster's stability. The only factor that matters is the duration of the loss of communication with master nodes—by default, after 40 seconds, a node enters the `Unreachable` state, and after 5 minutes, its pods are migrated to other nodes. Both thresholds are configured using the nodeMonitorGracePeriodSeconds and failedNodePodEvictionTimeoutSeconds parameters.

These delays affect the performance of everything running on the nodes—including the platform's own components. For components that access the Kubernetes API (DP operators, Prometheus, Ingress controllers), latency to the master nodes is added to every request. Latency between nodes is added to every network request between pods and to DNS lookups, and—when using replicated storage—to the time it takes to write to disk.

Thus, the acceptable value is determined by the requirements of the running workload. For clusters distributed across remote sites, this means that the platform will continue to operate, but application latencies will reflect the geographic distribution of the nodes.

Traffic between master nodes

Port	Protocol	Purpose
`2379, 2380`	TCP	etcd replication
`4200`	TCP	Cluster API webhook handler
`4201`	TCP	Webhook handler for VMware Cloud Director cloud provider
`4223`	TCP	Deckhouse controller webhook handler
`4281`	TCP	`registry` storage: replication of images between the replicas on master nodes, and reads by `registry-packages-proxy` on a master node that has no `registry-agent` yet
`4290`	TCP	Node controller webhook handler
`4291`	TCP	Node controller metrics
`4292`	TCP	Node controller health probe

Traffic from master nodes to regular nodes

Port	Protocol	Purpose
`22`	TCP	SSH for static node bootstrapping by the static provider
`10250`	TCP	kubelet
`4221`	TCP	`bashible apiserver` for delivering node configurations
`4227`	TCP	`runtime-audit-engine` webhook handler

Traffic from nodes to master nodes

Port	Protocol	Purpose
`4234`	UDP	NTP for time synchronization between nodes
`6443`	TCP	`kube-apiserver` for controllers working in the node's host network namespace
`4203`	TCP	`machine-controller-manager` metrics
`4219`	TCP	Proxy for registry packages `registry-packages-proxy`
`4222`	TCP	Deckhouse controller metrics
`4281`	TCP	`registry` storage: image pulls by the `registry-agent` of every node, and reads by the `terraform-manager` pods from the pod network (air-gapped clusters)
`4283`	TCP	`registry` storage: publishing images through the `registry-push` Ingress (`d8 mirror push`); reached from the nodes running the Ingress controller
`4284`	TCP	`registry` storage metrics, the 127.0.0.1 upstream of its kube-rbac-proxy. The port carries no traffic between nodes and needs no firewall rule; it is listed because the storage runs on the host network and occupies this port on every master node
`4285`	TCP	`registry` storage metrics
`4296`	TCP	`control-plane-manager` metrics
`4294`	TCP	`fencing-controller` metrics of the `node-manager` module. The controller uses the host network on the master nodes. On the same port, it serves its health probes on 127.0.0.1

Port	Protocol	Purpose
`5002, 5004, 5051`	TCP	`registry` storage on 127.0.0.1: registry metrics and debug (5002), the internal upstream of the registry (5004), the token service (5051). The ports carry no traffic between nodes and need no firewall rule; they are listed because the storage runs on the host network and occupies these ports on every master node

Traffic between nodes

Port	Protocol	Purpose
—	ICMP	ICMP for node-to-node connectivity monitoring
`4202`	TCP	`sds-node-configurator` node agent metrics
`4204`	TCP	Deckhouse controller debug
`4205`	TCP	`ebpf-exporter` module metrics
`4206`	TCP	`node-exporter` module metrics
`4207`	TCP	`ingress-nginx` controller metrics for the `HostWithFailover` inlet
`4208`	TCP	`ingress-nginx` controller metrics for the `HostWithFailover` inlet
`4209`	TCP	Kubernetes control plane metrics
`4210`	TCP	`kube-proxy` metrics
`4211`	TCP	Cluster API metrics
`4212`	TCP	`runtime-audit-engine` module metrics
`4213`	TCP	`kube-router` metrics

Port	Protocol	Purpose
`4214`	TCP	API of the `sds-replicated-volume` module agent
`4215`	TCP	`sds-replicated-volume` agent metrics
`4216`	TCP	`storage-status` agent metrics, the address the proxy forwards to
`4217`	TCP	`storage-status` agent metrics
`4218`	TCP/UDP	Synchronization of the `speaker` components in `metallb` modules via memberlist protocol
`4220`	TCP	Metrics of the `speaker` components in `metallb` modules
`4224`	TCP	`node-local-dns` metrics
`4225`	TCP/UDP	Synchronization of the `speaker` components in `metallb` modules via memberlist protocol
`4226`	TCP	Metrics of the `speaker` components in `metallb` modules
`4228`	TCP	`sds-node-configurator` agent healthcheck
`4229`	TCP	`csi-nfs` CSI controller healthcheck
`4230`	TCP	`csi-nfs` CSI node healthcheck
`4231`	TCP	`csi-hpe` CSI controller healthcheck
`4232`	TCP	`csi-hpe` CSI node healthcheck
`4233`	TCP	`user-authz` authorization webhook metrics

Port	Protocol	Purpose
`4235`	TCP	`csi-s3` CSI controller healthcheck
`4236`	TCP	`csi-s3` CSI node healthcheck
`4237`	TCP	`csi-scsi-generic` CSI controller healthcheck
`4238`	TCP	`csi-scsi-generic` CSI node healthcheck
`4239`	TCP	`storage-status` agent healthcheck
`4240`	TCP	CNI Cilium node-to-node healthcheck
`4241`	TCP	CNI Cilium agent metrics
`4242`	TCP	CNI Cilium operator metrics
`4243`	TCP	`user-authz` authorization webhook metrics, the 127.0.0.1 upstream of its kube-rbac-proxy. The port carries no traffic between nodes and needs no firewall rule; it is listed because the webhook runs on the host network and occupies this port on every master node
`4244`	TCP	`cilium-hubble` module API
`4245`	TCP	`chrony-exporter` metrics
`4246`	TCP	`csi-ceph` CephFS CSI controller healthcheck
`4247`	TCP	`csi-ceph` RBD CSI controller healthcheck
`4248`	TCP	`csi-yadro-tatlin-unified` CSI controller healthcheck
`4249`	TCP	`csi-yadro-tatlin-unified` CSI node healthcheck

Port	Protocol	Purpose
`4250`	TCP	`fencing-agent` metrics of the `node-manager` module. The agents use the host network. Thus they occupy this port on every cluster node when fencing is enabled on at least one node group. On the same port, the agent serves its health probes on 127.0.0.1
`4251`	TCP	`csi-hitachi` CSI node healthcheck
`4252`	TCP	`csi-ceph` RBD CSI node healthcheck
`4253`	TCP	`csi-ceph` CephFS CSI node healthcheck
`4254`	TCP	`csi-netapp` CSI controller healthcheck
`4255`	TCP	`csi-netapp` CSI node healthcheck
`4256`	TCP	`csi-netapp` CSI controller metrics
`4257`	TCP	`csi-netapp` CSI controller API port
`4258`	TCP	`csi-huawei` CSI controller webhook port
`4259`	TCP	`csi-huawei` CSI node healthcheck
`4260`	TCP	`csi-huawei` CSI controller metrics
`4261`	TCP	`ipv4-options-guard` metrics of the `cni-flannel` and `cni-simple-bridge` agents
`4262`	TCP	`sds-replicated-volume` CSI node healthcheck
`4263`	TCP	`service-with-healthchecks` module metrics

Port	Protocol	Purpose
`4264`	TCP	`node-local-dns` safe-updater healthcheck
`4265`	TCP	`node-local-dns` safe-updater pprof
`4266`	TCP	`csi-scsi-generic` CSI controller metrics
`4267`	TCP	`csi-scsi-generic` CSI node metrics
`4268`	TCP	`sds-replicated-volume` CSI node metrics
`4269`	TCP	`sds-replicated-volume` agent healthcheck
`4270`	TCP	`sds-replicated-volume` agent metrics, the address the proxy forwards to
`4273`	TCP	`sds-replicated-volume` DRBD metrics of the agent
`4274`	TCP	`sds-replicated-volume` DRBD metrics of the agent, the address the proxy forwards to
`4275`	TCP	`sdn` agent healthcheck
`4276`	TCP	`user-authz` Permission Browser API server metrics
`4278`	TCP	`csi-mind-ustor` CSI node healthcheck
`4279`	TCP	`csi-hitachi` CSI controller healthcheck
`4135-4199`	TCP	`virtualization` live migration tunnels between cluster nodes
`4280`	TCP	`virtualization` USB over IP (usb/ip) protocol for sharing USB devices between cluster nodes

Port	Protocol	Purpose
`4282`	TCP	Bootstrap endpoint for downloading `rpp-get` from `registry-packages-proxy`
`4286`	TCP	Istio CNI metrics. On 127.0.0.1 of every node the same port is taken by the `registry-agent` metrics, which carry no traffic between nodes and need no firewall rule
`4287`	UDP	WireGuard port for traffic encryption in CNI Cilium
`4288`	TCP	`monitoring-ping` metrics
`4289`	TCP	`monitoring-ping` metrics
`4293`	TCP	Healthcheck of the `opagent` agent of the `observability` module
`4294`	TCP	CNI Cilium ClusterMesh etcd on control-plane nodes, connected to by kvstoremesh on the control-plane nodes of the peer clusters
`4295-4297`	UDP	Used by the `cni-cilium` module for VXLAN encapsulation of inter-pod traffic in multiple nested virtualization — when DP with the `virtualization` module enabled is deployed inside virtual machines that are also created in DP with the `virtualization` module enabled
`4298`	UDP	Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods if the cluster was deployed on DP version starting from **1.71** (for clusters deployed on DP versions up to **1.71**, see the note for ports `4299/UDP`, `8469/UDP`, and `8472/UDP`)

Port	Protocol	Purpose
`4299`	UDP	**For clusters deployed on DP versions 1.64–1.70.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DP to newer versions will not change the port used unless the `virtualization` module is enabled. **Note** that in such clusters, enabling the `virtualization` module on DP up to version 1.70 changes the port to `4298/UDP`
`5001`	TCP	`registry-agent` on 127.0.0.1, through which the container runtime and Deckhouse pull images. The port carries no traffic between nodes and needs no firewall rule; it is listed because the agent runs on the host network and occupies this port on every node
`7000-7999`	TCP	`sds-replicated-volume` DRBD replication
`8469`	UDP	**For clusters deployed on DP version 1.63 and below with the `virtualization` module enabled prior to DP version 1.63.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DP to newer versions will not change the occupied port

Port	Protocol	Purpose
`8472`	UDP	**For clusters deployed on DP version 1.63 and below.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DP to newer versions will not change the occupied port if the `virtualization` module is not enabled. **Note** that in such clusters, enabling the `virtualization` module on DP before version 1.70 changes the port: • Enabling the `virtualization` module on DP version 1.63 and below will change it to `8469/UDP` and will not change with subsequent DP updates • Enabling the `virtualization` module on DP starting from version 1.64 will change it to `4298/UDP` and will not change with subsequent DP updates
`8500`	TCP/UDP	Gossip network of the `fencing-agent` components of the `node-manager` module (memberlist protocol). Open between the InternalIP addresses of all cluster nodes once fencing is enabled on at least one node group. If the second network of fencing (SystemNetwork) is set, also open UDP between the addresses of this network

Static `virtualization` ports on cluster nodes (`4100-4134/TCP` reserved)

Port	Protocol	Purpose
`4100`	TCP	`virtualization` `virt-handler` healthz and metrics
`4101`	TCP	`virtualization` `virt-handler` console server
`4102`	TCP	`virtualization` `virt-synchronization-controller` API that synchronizes the state of VM migrations between clusters. Used only with the `Peering` feature gate; the nodes of the other cluster connect to it

Port	Protocol	Purpose
`4103`	TCP	`virtualization virt-synchronization-controller` health check. Used only with the `Peering` feature gate
`4106`	TCP	`virtualization vm-route-forge` pprof in debug mode

External traffic to master nodes

Port	Protocol	Purpose
`22`	TCP	SSH for initializing Deckhouse Platform
`6443`	TCP	Direct access to the apiserver **Optional**. This port may be needed for direct access to `kube-apiserver` bypassing Ingress / Gateway API when publishing through a load balancer is not configured or unavailable due to infrastructure constraints

External traffic to frontend nodes

Port	Protocol	Purpose
`80, 443`	TCP	Application ports for requests to Ingress controllers over HTTP and HTTPS. Note that these ports are configurable in the IngressNginxController resource and may vary across installations
`5416`	UDP	OpenVPN **Optional**. Required only when the `openvpn` module is enabled
`5416`	TCP	OpenVPN **Optional**. Required only when the `openvpn` module is enabled

Port	Protocol	Purpose
`10256`	TCP	Healthcheck port for external balancers **Optional**. This is the Cilium healthz port checked by external load balancers, primarily cloud load balancers for `LoadBalancer` services. The port is not needed with `HostPort` / `HostPortWithProxyProtocol` inlets without an external load balancer
`30000-32767`	TCP	NodePort range **Optional**. Needed when `LoadBalancer` services are used. A cloud load balancer delivers traffic to nodes through a NodePort. The range is not needed with `HostPort` inlets

External traffic for all nodes

All ports in this section are used for outbound traffic from nodes.

Port	Protocol	Purpose
`53`	UDP	DNS
`53`	TCP	DNS
`123`	UDP	NTP for synchronization with external time servers
`179`	TCP	BGP peering with the infrastructure router for network traffic balancing (metallb module) **Optional**. Required only for the `metallb` module in BGP mode, from nodes running `speaker`. It is not required in L2 mode
`443`	TCP	Container registry

Additional resources

- Module `cni-cilium` parameters
- Module `cni-flannel` parameters
- Module `control-plane-manager` parameters

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The list of alerts

The page displays a list of all alerts of monitoring in the Deckhouse Platform.

> The list does not contain alerts of modules from a source.

Alerts are grouped by modules. To the right of the alert name, there are icons indicating the minimum DP edition in which the alert is available, and the alert severity.

For each alert, a summary is provided, and if available, the detailed alert description can be viewed by expanding it.

Alert severity

Alert descriptions contain the Severity (S) parameter, which indicates the level of criticality. Its value varies from `S1` to `S9` and can be interpreted as follows:

- `S1` — maximum level, critical failure/crash (immediate action required);
- `S2` — high level, close to maximum, possible accident (rapid response required);
- `S3` — medium level, potentially serious problem (verification required);
- `S4`-`S9` — low level. There is a problem, but overall performance is not impaired.

Module admission-policy-engine

- ControllerPolicyViolation CE S4
  ▼ **Pod-creating controllers in the XXX namespace violate the XXX policy, so their Pods are denied.**

  The `XXX` policy (XXX) denies the Pods that violate it. For a transition period, it only warns about a violation in a pod-creating controller (Deployment, StatefulSet, DaemonSet, Job, CronJob, ReplicationController) instead of denying the controller. The controller is accepted, but it cannot create its Pods: the denial appears only in the events of the ReplicaSet or the controller.

  Controllers that violate the policy in the namespace: VALUE.

  The `d8-pod-security-baseline` and `d8-pod-security-restricted` policies are the Pod Security Standards.

  To list the violating controllers and the rules they break, run the following Prometheus query:

  ```
  count by (violating_kind, violating_name, kind, violation_msg) (
    d8_gatekeeper_exporter_constraint_violations{
      source_type="XXX",
      name=~"d8-controllers-XXXXXX-deny(-d8)?(-default)?XXX",
      violating_namespace="XXX"
    }
  )
  ```

  Bring the pod template of each controller in line with the policy. To deny such controllers instead of warning about them, set `spec.controllerValidation.mode: Enforce` in the policy, or the `podSecurityStandards.controllerValidationEnforcement` module parameter to `Policy`. You can also check the admission-policy-engine reports in the Deckhouse web interface.

- D8AdmissionPolicyEngineAuditUnavailable CE S4
  ▼ **The audit controller of admission-policy-engine has no available replicas.**

  The `gatekeeper-audit` Deployment in the `d8-admission-policy-engine` namespace has no available replicas.

  Admission of new objects is not affected, but the module stops reporting and stops picking up new

policies:

- Violations among the objects that already run are not collected, so the `PodSecurityStandardsViolation`, `OperationPolicyViolation` and `SecurityPolicyViolation` alerts go silent instead of reporting that all is well.
- The list of resources the webhook intercepts is not updated, so a policy added while the audit is down is not enforced on admission.

Find out why the replica is gone:

```
d8 k -n d8-admission-policy-engine get pods -l app=gatekeeper,control-plane=audit-controller
d8 k -n d8-admission-policy-engine describe pods -l app=gatekeeper,control-plane=audit-controller
d8 k -n d8-admission-policy-engine logs deploy/gatekeeper-audit -c manager --tail=200
d8 k -n d8-admission-policy-engine logs deploy/gatekeeper-audit -c constraint-exporter --tail=200
```

- D8AdmissionPolicyEngineNotBootstrapped CE S7
  ▼ Not every ConstraintTemplate of the admission-policy-engine module is ready for 10 minutes.

  Some ConstraintTemplates of the admission-policy-engine module are missing or not processed by Gatekeeper. The constraints of a template are applied only once Gatekeeper has established its CRD.

  On a new installation, the module also renders no webhooks and no policies until every template is ready. In a cluster where the module has already worked, the webhooks and the constraints of the other templates stay in effect.

  Steps to troubleshoot:

  1. Verify that the module’s components are up and running:

     ```
     d8 k get pods -n d8-admission-policy-engine
     ```

  2. Check logs for issues, such as missing constraint templates or incomplete CRD creation:

     ```
     d8 k logs -n d8-system -lapp=deckhouse --tail=1000 | grep admission-policy-engine
     ```

- D8AdmissionPolicyEngineWebhookDegraded CE S3
  ▼ The admission webhook of admission-policy-engine has unavailable replicas.

  The `gatekeeper-controller-manager` Deployment in the `d8-admission-policy-engine` namespace has unavailable replicas, and the remaining ones still serve admission requests.

  The webhook has no redundancy left. Losing the last replica makes the API server reject every request the webhook intercepts, because the webhook has `failurePolicy: Fail`.

  A rollout of the Deployment raises this alert for a short time and needs no action. If the alert holds, find out why the replica does not start:

  ```
  d8 k -n d8-admission-policy-engine get pods -l app=gatekeeper,control-plane=controller-manager -o
  wide
  d8 k -n d8-admission-policy-engine describe pods -l app=gatekeeper,control-plane=controller-
  manager
  d8 k -n d8-admission-policy-engine get events --sort-by=.lastTimestamp | tail -30
  ```

- D8AdmissionPolicyEngineWebhookNotServing CE S2
  ▼ The API server cannot call the admission webhook of admission-policy-engine.

  The API server is rejecting requests because it fails to call a webhook of the `admission-policy-engine` module. The webhook has `failurePolicy: Fail`, so the cluster is blocked for everything the webhook intercepts.

  The alert is independent of the replica counters: it fires while the pods are `Ready` as well. A replica that is running but unreachable to the API server produces the same outage, and only this alert reports it. Typical causes are a `gatekeeper-webhook-server-cert` that no longer matches the `caBundle` of the ValidatingWebhookConfiguration, an empty `gatekeeper-webhook-service` endpoint list, and a network policy that blocks the API server.

  The API server counts a webhook that exceeds its `timeoutSeconds` of 15 the same way it counts one it cannot reach, so a few slow evaluations of a large object raise the metric as well. They do not raise the alert: `for: 15m` requires the failures to continue, while a burst leaves the rate

elevated for little more than its own duration plus the five-minute window.

Read the reason the API server reports, then check the certificate and the endpoints:

```
d8 k -n d8-admission-policy-engine logs deploy/gatekeeper-controller-manager -c manager --all-pods=true --tail=200
d8 k -n d8-admission-policy-engine get endpoints gatekeeper-webhook-service
d8 k get validatingwebhookconfiguration d8-admission-policy-engine-config -o jsonpath='{.webhooks[0].clientConfig.caBundle}' | base64 -d | openssl x509 -noout -subject -dates
d8 k -n d8-admission-policy-engine get secret gatekeeper-webhook-server-cert -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -subject -dates
```

The two certificates must match. Deckhouse Platform reissues them through the module’s hook, so deleting the secret makes it regenerate the pair.

- D8AdmissionPolicyEngineWebhookUnavailable CE S2
  ▾ The admission webhook of admission-policy-engine has no available replicas.

  The `gatekeeper-controller-manager` Deployment in the `d8-admission-policy-engine` namespace has no available replicas.

  The Deployment serves the admission webhook of the module, and the webhook has `failurePolicy: Fail`. While no replica is available, the API server rejects every request the webhook intercepts, cluster-wide:

  - Pods and the controllers that create them are not created or updated.
  - Deletion of those resources is rejected as well.
  - Roles, RoleBindings and Gatekeeper constraints are not applied.
  - The `d8 k exec`, `d8 k attach` and `d8 k debug` requests intercepted by the webhooks are rejected. This includes `d8 k exec` and `d8 k attach` in the `d8-*` and `kube-*` namespaces.

  Find out why the replicas are gone:

  ```
  d8 k -n d8-admission-policy-engine get deploy gatekeeper-controller-manager
  d8 k -n d8-admission-policy-engine get pods -l app=gatekeeper,control-plane=controller-manager -o wide
  d8 k -n d8-admission-policy-engine describe pods -l app=gatekeeper,control-plane=controller-manager
  d8 k -n d8-admission-policy-engine logs deploy/gatekeeper-controller-manager -c manager --all-pods=true --tail=200
  ```

  If a pod stays `Pending`, look at the events of the namespace and at the nodes the replicas can run on:

  ```
  d8 k -n d8-admission-policy-engine get events --sort-by=.lastTimestamp | tail -30
  d8 k get nodes -l node-role.deckhouse.io/system
  d8 k get nodes -l node-role.kubernetes.io/control-plane
  ```

- OperationPolicyViolation CE S3
  ▾ XXXObjects in the XXX namespaceXXXCluster-scoped objectsXXX violate the XXX OperationPolicy.

  Existing objects XXXin the `XXX` namespace XXXof cluster-scoped kinds XXXviolate the `XXX` OperationPolicy. An object is checked when it is created or updated, so these objects were created before the policy applied to them or before its rules changed.

  Objects that violate the policy: VALUE.

  To list the violating objects and the rules they break, run the following Prometheus query:

```
count by (violating_kind, violating_name, kind, violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    source_type="OperationPolicy",
    name="XXX",
    violation_enforcement="deny",
    violating_namespace="XXX"
  }
)
```

Bring the objects in line with the policy. You can also check the admission-policy-engine reports in the Deckhouse web interface.

- PodSecurityStandardsViolation CE S3
  ▼ Pods in the XXX namespace violate the XXX Pod Security Standard.

  Running Pods in the `XXX` namespace violate the Pod Security Standard published as the `XXX` SecurityPolicy. A Pod is checked when it is created, so these Pods were created before the standard applied to them or before its rules changed. They keep running until they are restarted, and are denied then.

  Pods that violate the standard in the namespace: VALUE.

  To list the violating Pods and the rules they break, run the following Prometheus query:

  ```
  count by (violating_name, kind, violation_msg) (
    d8_gatekeeper_exporter_constraint_violations{
      source_type="PSS",
      name=~"XXX-.+",
      violation_enforcement="deny",
      violating_kind="Pod",
      violating_namespace="XXX"
    }
  )
  ```

  Bring the workloads in line with the standard, or exempt them with a SecurityPolicyException. You can also check the admission-policy-engine reports in the Deckhouse web interface.

- SecurityPolicyViolation CE S3
  ▼ XXXObjects in the XXX namespaceXXXCluster-scoped objectsXXX violate the XXX SecurityPolicy.

  Existing objects XXXin the `XXX` namespace XXXof cluster-scoped kinds XXXviolate the `XXX` SecurityPolicy. An object is checked when it is created or updated, so these objects were created before the policy applied to them or before its rules changed.

  Objects that violate the policy: VALUE.

  To list the violating objects and the rules they break, run the following Prometheus query:

  ```
  count by (violating_kind, violating_name, kind, violation_msg) (
    d8_gatekeeper_exporter_constraint_violations{
      source_type="SecurityPolicy",
      name="XXX",
      violation_enforcement="deny",
      violating_namespace="XXX"
    }
  )
  ```

  Bring the objects in line with the policy, or exempt them with a SecurityPolicyException. You can also check the admission-policy-engine reports in the Deckhouse web interface.

Module cert-manager

- CertmanagerCertificateChallengePending CE S4
  ▼ ACME Challenge for domain "XXX" is pending for more than 1 hour.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than

1 hour and requires investigation.

Common causes:

- DNS misconfiguration (for DNS-01 challenges).
- Ingress or firewall issues blocking HTTP-01 validation.
- ACME provider rate limiting.
- Network connectivity issues.

Resolution steps:

1. Check the Challenge status and reason:

```
d8 k -n NAMESPACE describe challenge NAME
```

2. Check `cert-manager` logs for errors:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager --tail=100
```

3. If the challenge is stuck, delete the Order to trigger a retry:

```
d8 k -n NAMESPACE get orders
d8 k -n NAMESPACE delete order ORDER_NAME
```

- CertmanagerCertificateChallengePending CE S6
  ▼ ACME Challenge for domain "XXX" is pending for more than 10 minutes.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than 10 minutes.

  This may indicate issues with the ACME challenge validation process.

  Common causes:

  - DNS propagation delays (for DNS-01 challenges).
  - Ingress or firewall issues blocking HTTP-01 validation.
  - ACME provider rate limiting or temporary unavailability.

  Diagnostic steps:

  1. Check the Challenge status:

```
d8 k -n NAMESPACE describe challenge NAME
```

  2. Check `cert-manager` logs for errors:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager --tail=100
```

- CertmanagerCertificateChallengeStuck CE S4
  ▼ ACME Challenge for domain "XXX" is stuck in "STATUS_REFERENCE" state.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` is in `STATUS_REFERENCE` state and requires manual intervention.

  Reason: XXX

  The certificate will not be issued or renewed until the issue is resolved.

  Common causes:

  - DNS misconfiguration (for DNS-01 challenges).
  - Ingress or firewall issues (for HTTP-01 challenges).
  - Rate limiting by the ACME provider.
  - Authorization expired before challenge completion.

  Resolution steps:

  1. Check the Challenge status and reason:

```
d8 k -n NAMESPACE describe challenge NAME
```

  2. Delete the failed Order to trigger a new certificate request:

```
d8 k -n NAMESPACE get orders
d8 k -n NAMESPACE delete order ORDER_NAME
```

- CertmanagerCertificateExpired CE S4
  ▼ **Certificate NAMESPACE/NAME is not provided.**

  Certificate is not provided.

  To check the certificate details, run the following command:

  ```
  d8 k -n NAMESPACE describe certificate NAME
  ```

- CertmanagerCertificateExpiredSoon CE S4
  ▼ **Certificate will expire soon.**

  The certificate `NAMESPACE/NAME` will expire in less than two weeks.

  To check the certificate details, run the following command:

  ```
  d8 k -n NAMESPACE describe certificate CERTIFICATE-NAME
  ```

- CertmanagerCertificateOrderErrors CE S5
  ▼ **Cert-manager couldn't order a certificate.**

  Cert-manager received responses with the status code `STATUS_REFERENCE` when requesting `SCHEME://HOST_NAMEPATH`.

  This can affect certificate ordering and prolongation in the future. For details, check the cert-manager logs using the following command:

  ```
  d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager
  ```

Module chrony

- NodeTimeNotSynchronizedAtBoot CE S5
  ▼ **Master node NODE_NAME started kubelet before its clock was synchronized.**

  During the last boot of the master node `NODE_NAME`, the clock was not synchronized from NTP before kubelet started. Either no NTP server answered within the waiting time, the servers that answered disagreed with each other, nodelet could not run the synchronization, or the synchronization hung and systemd stopped it. kubelet and the control plane started with a clock taken from the RTC or the hypervisor. If that clock was wrong, the control plane could issue tokens and certificates with incorrect timestamps.

  To resolve the issue:

  1. Check the synchronization log on the node. On a node managed by bashible, run:

     ```
     journalctl -u nodelet.service -b
     ```

     A Deckhouse Engine node has no SSH access. Read the log from a debug pod, where `IMAGE` is any image available to the cluster that has the `chroot` utility:

     ```
     d8 k -n kube-system debug node/NODE_NAME -i --profile=sysadmin --image=IMAGE -- chroot
     /host journalctl --no-pager -u nodelet.service -b | grep -E 'clock|NTP'
     ```

  2. On a node managed by bashible, check that the installed `nodelet.service` has `Type=notify` and that nodelet started. If the package is missing or outdated, bashible installs it on its next run. To start that run sooner, run the following command on the node:

     ```
     rm -f /var/lib/bashible/configuration_checksum
     ```

  3. Otherwise, ensure the NTP servers defined in the chrony configuration are reachable through the node network and their names are resolvable by the node DNS settings, without Kubernetes DNS. If the log reports `the NTP servers disagree` or `the addresses of SERVER disagree`, use servers whose answers agree with each other.

  4. The alert stays active until the node boots again and the synchronization succeeds. On a

node managed by bashible, disabling boot-time synchronization also clears it. After you have checked the clock on the node, you can clear the alert without a reboot. On a node managed by bashible, run on the node:

```
systemctl reset-failed nodelet.service
rm -f /var/run/node-exporter-textfile/d8_early_time_sync.prom
```

If `nodelet.service` is in the `failed` state, `systemctl reset-failed` keeps bashible from writing the failure metric again.

On a Deckhouse Engine node, remove the file from a debug pod, as in step 1:

```
d8 k -n kube-system debug node/NODE_NAME -i --profile=sysadmin --image=IMAGE -- chroot
/host rm -f /var/run/node-exporter-textfile/d8_early_time_sync.prom
```

Each debug command leaves a `node-debugger-*` pod in the `kube-system` namespace. Delete it with `d8 k -n kube-system delete pod POD_NAME`, using the name the command printed.

The alert resolves about 10 minutes after the file is removed.

- NodeTimeOutOfSync CE S5
  ▾ **Clock on the node NODE_NAME is drifting.**

  Time on the node `NODE_NAME` is out of sync and drifts apart from the NTP server clock by VALUE seconds.

  To resolve the time synchronization issues:

  - Fix network errors:
    - Ensure the upstream time synchronization servers defined in the chrony configuration are available.
    - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
  - Modify the NTP servers list defined in the chrony configuration.
- NTPDaemonOnNodeDoesNotSynchronizeTime CE S5
  ▾ **NTP daemon on the node NODE_NAME haven't synchronized time for too long.**

  Steps to troubleshoot:

  1. Check if the chrony pod is running on the node by executing the following command:

     ```
     d8 k -n d8-chrony --field-selector spec.nodeName="NODE_NAME" get pods
     ```

  2. Verify the chrony daemon’s status by executing the following command:

     ```
     d8 k -n d8-chrony exec POD_NAME -- /opt/chrony-static/bin/chronyc sources
     ```

  3. Resolve the time synchronization issues:

     - Fix network errors:
       - Ensure the upstream time synchronization servers defined in the chrony configuration are available.
       - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
     - Modify the NTP servers list defined in the chrony configuration.

Module cloud-provider-aws

- D8CloudProviderAWSIMDSv2NotConverged CE S6
  ▾ **Some EC2 instances of the cluster still allow IMDSv1.**

  `imdsv2: true` is set in `AWSClusterConfiguration`, but VALUE EC2 instances of the cluster still accept metadata requests without a session token, which leaves them more exposed to unauthorized metadata access, including through SSRF attacks.

  Existing instances are not reconfigured by the setting alone:

  1. Master, static and bastion instances are updated by a converge run. Start the Deckhouse Platform installer and run `dhctl converge`.
  2. Ephemeral instances are re-created when their nodes are rolled, which happens on its own.

Check that the nodes of every `CloudEphemeral` NodeGroup are being replaced: `d8 k get nodegroups`.

The instance state is re-read once per discovery period of the cloud data discoverer (one hour by default), so the alert can lag behind the last node being replaced.

Module cloud-provider-huaweicloud

- D8HuaweiCloudMachineStuckInDeleting EE S6
  ▼ HuaweiCloudMachine NAME is stuck in Deleting for more than 24h.

  The HuaweiCloudMachine `NAME` in namespace `NAMESPACE` has been stuck in Deleting state for more than 24 hours.

  This usually means the deletion job in HuaweiCloud is failing or the API is unavailable.

  To inspect the machine status, run:

  ```
  d8 k get huaweicloudmachine -n NAMESPACE NAME -o yaml
  ```

  Check the controller logs for errors:

  ```
  d8 k logs -n d8-cloud-provider-huaweicloud -l app=caphc-controller-manager
  ```

Module cloud-provider-openstack

- OpenStackPreemptibleUnsupported EE S8
  ▼ NodeGroup NODE_GROUP_NAME requests preemptible instances, but this cluster cannot deliver them (reason: XXX).

  The OpenStackInstanceClass referenced by NodeGroup `NODE_GROUP_NAME` has `spec.preemptible: true`, but the raw Nova `preemptible` tag will not reach the virtual machines: nodes are created as regular (non-preemptible) VMs, and Selectel-side preemption never kicks in.

  XXX **Reason:** the NodeGroup runs on the **MCM engine**, which has no way to emit raw Nova tags — `machine-controller-manager`'s OpenStackMachineClass CRD does not expose the field. Only the CAPI engine (`capi-controller-manager` + `capo-controller-manager`) renders raw Nova tags via `OpenStackMachineTemplate.spec.template.spec.tags`.

  How to fix:

  1. Confirm the current engine and offending InstanceClass:

     ```
     d8 k get ng NODE_GROUP_NAME -o jsonpath='{.status.engine}{"\n"}'
     d8 k get ng NODE_GROUP_NAME -o jsonpath='{.spec.cloudInstances.classReference}
     {"\n"}'
     ```

  2. Choose one of the following actions:

     - **Recommended.** Remove the `node.deckhouse.io/use-mcm` annotation from the NodeGroup so it migrates to CAPI. Nodes will be recreated on the CAPI engine, this time carrying the `preemptible` tag:

       ```
       d8 k annotate ng NODE_GROUP_NAME node.deckhouse.io/use-mcm-
       ```

     - Alternatively, if this NodeGroup must stay on MCM, set `spec.preemptible: false` on the OpenStackInstanceClass (nodes stay regular, the alert clears):

       ```
       d8 k patch openstackinstanceclass CLASS_NAME --type=merge -p '{"spec":
       {"preemptible":false}}'
       ```

       Note: a single OpenStackInstanceClass shared between a CAPI NodeGroup (where the field works) and an MCM one (where it doesn't) will keep firing this alert on the MCM side. Use separate InstanceClasses per engine when the classes differ in `preemptible`.

  XXX **Reason:** the cluster's `connection.authURL` does not point at Selectel (`selcloud.ru` / `selectel`). The raw Nova `preemptible` tag is a **Selectel-specific mechanism**; on other OpenStack providers Nova accepts the tag but ignores it, so instances are billed and behave as

regular non-preemptible VMs.

How to fix:

1. Confirm the offending InstanceClass and the cluster's authURL:

   ```
   d8 k get ng NODE_GROUP_NAME -o jsonpath='{.spec.cloudInstances.classReference}
   {"\n"}'
   d8 k -n kube-system get secret d8-node-manager-cloud-provider -o
   jsonpath='{.data.openstack}' | base64 -d | jq -r '.connection.authURL'
   ```

2. Set `spec.preemptible: false` on the OpenStackInstanceClass — the field only makes sense on Selectel-hosted clusters:

   ```
   d8 k patch openstackinstanceclass CLASS_NAME --type=merge -p '{"spec":
   {"preemptible":false}}'
   ```

   The alert clears on the next scrape after the change; existing nodes stay running (they are already regular VMs — the tag was never emitted).

XXX **Reason:** unknown (`XXX`). This alert should surface only two well-known cases (`mcm`, `non-selectel`). A new reason implies node-controller has been extended without this runbook — check the release notes for `cloud-provider-openstack` before deciding what to do. XXX

Module cloud-provider-yandex

- D8CloudProviderYandexCloudMigrationPending CE S6
  ▼ **The cloud-provider-yandex module is waiting for migration resources to be applied.**

  The `cloud-provider-yandex` module has detected the legacy `YandexProviderClusterConfiguration` secret in the cluster and is waiting for the new migration resources (ModuleConfig v2, CredentialSecret, NodeGroups, InstanceClasses) to be applied before completing the migration.

  To complete the migration, apply the resources from the `d8-migration-resources` Secret:

  ```
  d8 k -n d8-cloud-provider-yandex get secret d8-migration-resources -o
  jsonpath='{.data.resources\.yaml}' | base64 -d | d8 k apply -f -
  ```

  After applying the resources, the alert will disappear automatically once the module detects that all required resources are in place.

  After the migration is complete, the legacy `YandexProviderClusterConfiguration` secret is no longer needed and can be deleted manually:

  1. Make sure the cloud discovery data is stored outside the legacy secret. The module reads it from the `d8-candi-cloud-provider-discovery-data` Secret first and uses the legacy secret only as a fallback, so the following command must print a non-empty value:

     ```
     d8 k -n d8-cloud-provider-yandex get secret d8-candi-cloud-provider-discovery-data -o
     jsonpath='{.data.cloud-provider-discovery-data\.json}'
     ```

     If it prints nothing, the discovery data exists only in the legacy secret, and deleting that secret would lose it. Copy the payload over — the key holds exactly the same discovery data dhctl recorded from the infrastructure run:

     ```
     d8 k -n kube-system get secret d8-provider-cluster-configuration \
       -o jsonpath='{.data.cloud-provider-discovery-data\.json}' | base64 -d >
     /tmp/yandex-discovery.json

     d8 k -n d8-cloud-provider-yandex create secret generic d8-candi-cloud-provider-discovery-data
     \
       --from-file=cloud-provider-discovery-data.json=/tmp/yandex-discovery.json \
       --dry-run=client -o yaml | d8 k apply -f -
     ```

  2. Back up the secret. It contains cloud credentials and discovery data, so store the file securely:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration -o yaml > d8-provider-cluster-configuration-backup.yaml
```

3. Allow the deletion. The secret is protected by an admission webhook and cannot be deleted without the annotation:

```
d8 k -n kube-system annotate secret d8-provider-cluster-configuration deckhouse.io/allow-delete=true
```

4. Delete the secret:

```
d8 k -n kube-system delete secret d8-provider-cluster-configuration
```

- D8YandexNatInstanceConnectionsQuotaUtilization CE S4
  ▼ **Connection quota utilization of the Yandex NAT instance exceeds 85% over the last 5 minutes.**

  The connection quota for the Yandex NAT instance has exceeded 85% utilization over the past 5 minutes.

  To prevent potential issues, contact Yandex technical support and request an increase in the connection quota.

- NATInstanceWithDeprecatedAvailabilityZone CE S9
  ▼ **NAT instance NAME is in a deprecated availability zone.**

  The NAT instance `NAME` is located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud. To resolve this issue, migrate the NAT instance to either `ru-central1-a` or `ru-central1-b` by following the instructions below.

  > **i** The migration process involves irreversible changes and may result in a significant downtime. The duration typically depends on Yandex Cloud's response time and can last several tens of minutes.

  1. Migrate the NAT instance. To get `providerClusterConfiguration.withNATInstance`, run the following command:

     ```
     d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values -g cloud-provider-yandex -o json | jq -c | jq '.cloudProviderYandex.internal.providerClusterConfiguration.withNATInstance'
     ```

     - If you specified `withNATInstance.natInstanceInternalAddress` and/or `withNATInstance.internalSubnetID` in `providerClusterConfiguration`, remove them using the following command:

       ```
       d8 system edit provider-cluster-configuration
       ```

     - If you specified `withNATInstance.externalSubnetID` and/or `withNATInstance.natInstanceExternalAddress` in `providerClusterConfiguration`, change them to the appropriate values.

       To get the address and subnet ID, use the Yandex Cloud Console or CLI.

       To change `withNATInstance.externalSubnetID` and `withNATInstance.natInstanceExternalAddress`, run the following command:

       ```
       d8 system edit provider-cluster-configuration
       ```

  2. Run the appropriate edition and version of the Deckhouse installer container on the **local** machine. You may have to change the container registry address to do that. After that, perform the converge.

     - Get the appropriate edition and version of Deckhouse:

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr
'[:upper:]' '[:lower:]')
echo "DH_VERSION=$DH_VERSION DH_EDITION=$DH_EDITION"
```

- Run the installer:

```
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/"
registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

- Perform the converge:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/SSH_KEY_FILENAME --ssh-
user=USERNAME --ssh-host MASTER-NODE-0-HOST
```

3. Update the route table.
   - Get the route table name:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller
module values -g cloud-provider-yandex -o json | jq -c | jq '.global.prefix //
.global.clusterConfiguration.cloud.prefix'
```

   - Get the NAT instance name:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller
module values -g cloud-provider-yandex -o json | jq -c | jq
'.cloudProviderYandex.internal.providerDiscoveryData.natInstanceName'
```

   - Get the NAT instance internal IP address:

```
yc compute instance list | grep -e "INTERNAL IP" -e
NAT_INSTANCE_NAME_FROM_PREVIOUS_STEP
```

   - Update the route:

```
yc vpc route-table update --name ROUTE_TABLE_NAME_FROM_PREVIOUS_STEP --route
"destination=0.0.0.0/0,next-
hop=NAT_INSTANCE_INTERNAL_IP_FROM_PREVIOUS_STEP"
```

- NodeGroupNodeWithDeprecatedAvailabilityZone CE S9
  ▼ **NodeGroup NODE_GROUP_NAME contains nodes in a deprecated availability zone.**

  Certain nodes in the node group `NODE_GROUP_NAME` are located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud.

  Steps to troubleshoot:

  1. Identify the nodes that need to be migrated by running the following command:

```
d8 k get node -l "topology.kubernetes.io/zone=ru-central1-c"
```

  2. Migrate your nodes, disks, and load balancers to one of the supported zones: `ru-central1-a`, `ru-central1-b`, or `ru-central1-d`. Refer to the Yandex migration guide for detailed instructions.

> **i** You can’t migrate public IP addresses between zones. For details, refer to the migration guide.

Module cloud-provider-zvirt

- D8CloudProviderZvirtMigrationPending SE-PLUS S6
  ▼ The cloud-provider-zvirt module is waiting for migration resources to be applied.

  The `cloud-provider-zvirt` module has detected the legacy `ZvirtClusterConfiguration` secret in the cluster and is waiting for the new migration resources (ModuleConfig v2, CredentialSecret, NodeGroups, InstanceClasses) to be applied before completing the migration.

  To complete the migration, apply the resources from the `d8-migration-resources` Secret:

  ```
  d8 k -n d8-cloud-provider-zvirt get secret d8-migration-resources -o
  jsonpath='{.data.resources\.yaml}' | base64 -d | d8 k apply -f -
  ```

  After applying the resources, the alert will disappear automatically once the module detects that all required resources are in place.

  After the migration is complete, the legacy `ZvirtProviderClusterConfiguration` secret is no longer needed and can be deleted manually:

  1. Make sure the cloud discovery data is stored outside the legacy secret. The module reads it from the `d8-candi-cloud-provider-discovery-data` Secret first and uses the legacy secret only as a fallback, so the following command must print a non-empty value:

     ```
     d8 k -n d8-cloud-provider-zvirt get secret d8-candi-cloud-provider-discovery-data -o
     jsonpath='{.data.cloud-provider-discovery-data\.json}'
     ```

     If it prints nothing, the discovery data exists only in the legacy secret, and deleting that secret would lose it. Copy the payload over — the key holds exactly the same discovery data dhctl recorded from the infrastructure run:

     ```
     d8 k -n kube-system get secret d8-provider-cluster-configuration \
       -o jsonpath='{.data.cloud-provider-discovery-data\.json}' | base64 -d >
     /tmp/zvirt-discovery.json

     d8 k -n d8-cloud-provider-zvirt create secret generic d8-candi-cloud-provider-discovery-data
     \
       --from-file=cloud-provider-discovery-data.json=/tmp/zvirt-discovery.json \
       --dry-run=client -o yaml | d8 k apply -f -
     ```

  2. Back up the secret. It contains cloud credentials and discovery data, so store the file securely:

     ```
     d8 k -n kube-system get secret d8-provider-cluster-configuration -o yaml > d8-provider-
     cluster-configuration-backup.yaml
     ```

  3. Allow the deletion. The secret is protected by an admission webhook and cannot be deleted without the annotation:

     ```
     d8 k -n kube-system annotate secret d8-provider-cluster-configuration deckhouse.io/allow-
     delete=true
     ```

  4. Delete the secret:

     ```
     d8 k -n kube-system delete secret d8-provider-cluster-configuration
     ```

Module cni-cilium

- CiliumAgentEndpointsNotReady CE S4
  ▼ Over 50% of all known endpoints aren't ready in agent NAMESPACE/POD_NAME.

  For details, refer to the logs of the agent:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CiliumAgentMapPressureCritical CE S4
  ▾ **eBPF map MAP_NAME exceeds 90% utilization in agent NAMESPACE/POD_NAME.**

  The eBPF map resource utilization limit has almost been reached.

  Check with the vendor for potential remediation steps.

- CiliumAgentMetricNotFound CE S4
  ▾ **Agent NAMESPACE/POD_NAME isn't sending some metrics.**

  Steps to troubleshoot:

  1. Check the logs of the agent:

     ```
     d8 k -n NAMESPACE logs POD_NAME
     ```

  2. Verify the agent's health status:

     ```
     d8 k -n NAMESPACE exec -ti POD_NAME cilium-health status
     ```

  3. Compare the metrics with those of a neighboring agent.

  > **i** Note that the absence of metrics can indirectly indicate that new pods can't be created on the node due to connectivity issues with the agent.

- CiliumAgentPolicyImportErrors CE S4
  ▾ **Agent NAMESPACE/POD_NAME fails to import policies.**

  For details, refer to the logs of the agent:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CiliumAgentUnreachableHealthEndpoints CE S4
  ▾ **Agent NAMESPACE/POD_NAME can't reach some of the node health endpoints.**

  For details, query Cilium health status API:

  ```
  d8 k -n NAMESPACE exec POD_NAME -- cilium-health status
  ```

  You can also refer to the agent logs:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CniCiliumNetworkPeeringWithoutPeeringModule EE S4
  ▾ **NetworkPeering is enabled, but the peering module is disabled.**

  NetworkPeering connects the clusters peered by the `peering` module. With the module disabled, no new mesh with a peer comes up, and the meshes already configured are not updated.

  To resolve this issue, enable the `peering` module, or set `networkPeering.enabled: false` in the `cni-cilium` ModuleConfig.

- CniCiliumNonStandardVXLANPortFound CE S4
  ▾ **Cilium configuration uses a non-standard VXLAN port.**

  The Cilium configuration specifies a non-standard VXLAN port `XXX`. The recommended port number is `XXX`.

  To resolve this issue, update the `tunnel-port` parameter in the `cilium-configmap` ConfigMap located in the `d8-cni-cilium` namespace to match the recommended port.

  If you configured the non-standard port on purpose, ignore this alert.

- CniCiliumOrphanEgressGatewayPolicyFound SE-PLUS S4
  ▾ **Orphaned EgressGatewayPolicy with an irrelevant EgressGateway name has been found.**

  The cluster contains an orphaned EgressGatewayPolicy named `NAME` with an irrelevant EgressGateway name.

To resolve this issue, verify the EgressGateway name specified in the EgressGatewayPolicy resource `EGRESSGATEWAY` and update it as needed.

- CNICiliumPolicyWithRequiresFieldsFound CE S4
  ▼ The XXX XXXNAMESPACE/XXXNAME uses the fromRequires or toRequires fields.

  The XXX `NAME` XXX in the `NAMESPACE` namespaceXXX uses the `fromRequires` or `toRequires` fields.

  Cilium 1.19 and later doesn't support these fields. The Cilium agent of these versions can't read such policies and fails on all nodes. Deckhouse won't deploy a release with such a Cilium version until all these policies are rewritten.

  To resolve this issue, rewrite each `fromRequires` constraint as an `ingressDeny` rule with an inverted selector. Don't move the selector into the deny rule as is: this inverts the policy. A deny rule wins over allow rules of any policy, so it keeps the original scope: `fromRequires` applied to all policies selecting the endpoint.

  For example, `fromRequires` with `matchLabels: {env: prod}` becomes:

  ```
  ingressDeny:
  - fromEndpoints:
    - matchExpressions:
      - {key: env, operator: NotIn, values: [prod]}
  XXX
    - matchExpressions:
      - {key: k8s:io.kubernetes.pod.namespace, operator: Exists}
      - {key: k8s:io.kubernetes.pod.namespace, operator: NotIn, values: [NAMESPACE]}
  XXX
  ```

  - Add a separate `NotIn` selector for each label of the constraint. XXX
  - Keep the second selector: in a CiliumNetworkPolicy, `fromRequires` also implied the policy namespace, so sources from other namespaces were rejected. The `Exists` condition is required, otherwise the rule also blocks the host, the world, and other sources without a namespace. XXX
  - Rewrite `toRequires` the same way, as an `egressDeny` rule with `toEndpoints`.
  - Add the deny rule first, check the traffic, and only then remove `fromRequires` and `toRequires`.

  To list all affected policies, use the `d8_cni_cilium_policy_with_requires_fields == 1` expression in Prometheus.

- D8IPv4OptionsGuardAuditMode CE S9
  ▼ The CVE-2026-98432 filter only counts packets on VALUE nodes of the cluster.

  The `ipv4OptionsGuard.mode` of the CNI module is `Audit` on VALUE nodes: packets with IP options are only counted, not dropped, and MPTCP stays enabled. The nodes remain exposed to CVE-2026-98432.

  Clusters that existed before the mitigation get `Audit` by default. Once the `D8IPv4OptionsGuardMatchedTraffic` alert shows that no workload relies on IP options, switch to `Enforce` in the ModuleConfig of the CNI module (`cni-cilium`, `cni-flannel` or `cni-simple-bridge`):

  ```
  spec:
    settings:
      ipv4OptionsGuard:
        mode: Enforce
  ```

- D8IPv4OptionsGuardBackendMismatch CE S6
  ▼ The CVE-2026-98432 filter on the node NODE_NAME uses a different iptables mode than iptables-wrapper selected.

  The node `NODE_NAME` is protected, but the filter uses `XXX`: the kernel modules of the mode selected by iptables-wrapper are not loaded, while those of the other mode are.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The filter and the other Deckhouse components use different packet filtering frameworks on this node. Such a security configuration is harder to audit and troubleshoot.

Possible actions:

- Find out why the kernel modules of the mode selected by iptables-wrapper are not loaded on the node.
- If loading these kernel modules is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.

- D8IPv4OptionsGuardFilterBypassed CE S6
  ▼ **Packets can skip the CVE-2026-98432 filter in VALUE network namespaces on the node NODE_NAME.**

  The node `NODE_NAME` uses legacy iptables. In VALUE network namespaces, a rule in the `PREROUTING` chain of the `raw` table ahead of the jump to `D8-IPV4OPT-GUARD` can end the traversal of the table (for example, `-j ACCEPT`). Packets accepted by that rule never reach the filter.

  The guard puts its jump first when it installs the filter, but does not move it later, so as not to fight with other software on the node that manages its own rules. The affected network namespaces are named in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  Possible actions:

  - Find the software that inserted the rule (`iptables-legacy -t raw -S PREROUTING` in the network namespace) and make it append the rule instead, or use a target that does not end the traversal.
  - Or put the jump first by hand: `iptables-legacy -t raw -I PREROUTING 1 -j D8-IPV4OPT-GUARD`. The guard removes the old jump on its next pass.
- D8IPv4OptionsGuardMatchedTraffic CE S7
  ▼ **IPv4 packets with IP options were seen on the node NODE_NAME in the Audit mode.**

  Over the last hour, the ipv4-options-guard sidecar matched VALUE IPv4 packets with IP options (rule `XXX`) on the node `NODE_NAME`.

  In the `Audit` mode they are only counted. In the `Enforce` mode they would be dropped, and this alert is not raised.

  Legitimate TCP traffic almost never carries IP options. If a workload relies on them (for example, CIPSO labeled networking), exclude the node with the `ipv4-options-guard.deckhouse.io/disabled: "true"` label before switching `ipv4OptionsGuard.mode` to `Enforce`. Otherwise, this may be an attempt to exploit CVE-2026-98432.

- D8IPv4OptionsGuardMetricsAbsent CE S4
  ▼ **The CVE-2026-98432 filter state is unknown on the node NODE_NAME.**

  The CNI agent pod runs on the node `NODE_NAME`, but Prometheus gets no metrics from its `ipv4-options-guard` container.

  Without these metrics, the other `D8IPv4OptionsGuard*` alerts cannot fire for this node, so it may be unprotected unnoticed.

  Check the `ipv4-options-guard` container of the CNI agent pod on that node and the scraping of its metrics.

- D8IPv4OptionsGuardNodeUnprotected CE S4
  ▼ **The CVE-2026-98432 filter cannot be installed on the node NODE_NAME.**

  The ipv4-options-guard sidecar of the CNI agent could not protect the node `NODE_NAME` (reason: `XXX`).

  - `iptables_wrapper_failed` — iptables-wrapper could not determine the iptables mode of the node.
  - `kernel_modules_not_loaded` — the kernel modules of the mode selected by iptables-wrapper are not loaded, and loading them is not allowed.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The node remains exposed to CVE-2026-98432 until its kernel is updated.

  Possible actions:

  - Update the node kernel to a version with the fix and reboot the node.
  - If the node uses legacy iptables and loading the kernel modules named in the reason is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.
- D8IPv4OptionsGuardReconcileFailed CE S6
  ▼ **The CVE-2026-98432 filter failed in VALUE network namespaces on the node**

NODE_NAME.

Some network namespaces on the node `NODE_NAME` are not protected.

Check the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

- D8NodeHasUnmetKernelRequirements CE S4
  ▼ Nodes have unmet kernel requirements.

  Deckhouse has detected that some nodes don’t meet the required kernel constraints. As a result, certain modules can’t run on these nodes.

  Kernel requirements for each module:

  - **Cilium module**: Kernel version must be **>= 5.8**.

  To list all affected nodes, use the `d8_node_kernel_does_not_satisfy_requirements == 1` expression in Prometheus.

Module cni-flannel

- D8IPv4OptionsGuardAuditMode CE S9
  ▼ The CVE-2026-98432 filter only counts packets on VALUE nodes of the cluster.

  The `ipv4OptionsGuard.mode` of the CNI module is `Audit` on VALUE nodes: packets with IP options are only counted, not dropped, and MPTCP stays enabled. The nodes remain exposed to CVE-2026-98432.

  Clusters that existed before the mitigation get `Audit` by default. Once the `D8IPv4OptionsGuardMatchedTraffic` alert shows that no workload relies on IP options, switch to `Enforce` in the ModuleConfig of the CNI module (`cni-cilium`, `cni-flannel` or `cni-simple-bridge`):

  ```
  spec:
    settings:
      ipv4OptionsGuard:
        mode: Enforce
  ```

- D8IPv4OptionsGuardBackendMismatch CE S6
  ▼ The CVE-2026-98432 filter on the node NODE_NAME uses a different iptables mode than iptables-wrapper selected.

  The node `NODE_NAME` is protected, but the filter uses `XXX`: the kernel modules of the mode selected by iptables-wrapper are not loaded, while those of the other mode are.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The filter and the other Deckhouse components use different packet filtering frameworks on this node. Such a security configuration is harder to audit and troubleshoot.

  Possible actions:

  - Find out why the kernel modules of the mode selected by iptables-wrapper are not loaded on the node.
  - If loading these kernel modules is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.
- D8IPv4OptionsGuardFilterBypassed CE S6
  ▼ Packets can skip the CVE-2026-98432 filter in VALUE network namespaces on the node NODE_NAME.

  The node `NODE_NAME` uses legacy iptables. In VALUE network namespaces, a rule in the `PREROUTING` chain of the `raw` table ahead of the jump to `D8-IPV4OPT-GUARD` can end the traversal of the table (for example, `-j ACCEPT`). Packets accepted by that rule never reach the filter.

  The guard puts its jump first when it installs the filter, but does not move it later, so as not to fight with other software on the node that manages its own rules. The affected network namespaces are named in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  Possible actions:

  - Find the software that inserted the rule (`iptables-legacy -t raw -S PREROUTING` in the network namespace) and make it append the rule instead, or use a target that does not end the traversal.

    - Or put the jump first by hand: `iptables-legacy -t raw -I PREROUTING 1 -j D8-IPV4OPT-GUARD`. The guard removes the old jump on its next pass.
- D8IPv4OptionsGuardMatchedTraffic CE S7
  ▾ **IPv4 packets with IP options were seen on the node NODE_NAME in the Audit mode.**

  Over the last hour, the ipv4-options-guard sidecar matched VALUE IPv4 packets with IP options (rule `XXX`) on the node `NODE_NAME`.

  In the `Audit` mode they are only counted. In the `Enforce` mode they would be dropped, and this alert is not raised.

  Legitimate TCP traffic almost never carries IP options. If a workload relies on them (for example, CIPSO labeled networking), exclude the node with the `ipv4-options-guard.deckhouse.io/disabled: "true"` label before switching `ipv4OptionsGuard.mode` to `Enforce`. Otherwise, this may be an attempt to exploit CVE-2026-98432.

- D8IPv4OptionsGuardMetricsAbsent CE S4
  ▾ **The CVE-2026-98432 filter state is unknown on the node NODE_NAME.**

  The CNI agent pod runs on the node `NODE_NAME`, but Prometheus gets no metrics from its `ipv4-options-guard` container.

  Without these metrics, the other `D8IPv4OptionsGuard*` alerts cannot fire for this node, so it may be unprotected unnoticed.

  Check the `ipv4-options-guard` container of the CNI agent pod on that node and the scraping of its metrics.

- D8IPv4OptionsGuardNodeUnprotected CE S4
  ▾ **The CVE-2026-98432 filter cannot be installed on the node NODE_NAME.**

  The ipv4-options-guard sidecar of the CNI agent could not protect the node `NODE_NAME` (reason: `XXX`).

    - `iptables_wrapper_failed` — iptables-wrapper could not determine the iptables mode of the node.
    - `kernel_modules_not_loaded` — the kernel modules of the mode selected by iptables-wrapper are not loaded, and loading them is not allowed.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The node remains exposed to CVE-2026-98432 until its kernel is updated.

  Possible actions:

    - Update the node kernel to a version with the fix and reboot the node.
    - If the node uses legacy iptables and loading the kernel modules named in the reason is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.
- D8IPv4OptionsGuardReconcileFailed CE S6
  ▾ **The CVE-2026-98432 filter failed in VALUE network namespaces on the node NODE_NAME.**

  Some network namespaces on the node `NODE_NAME` are not protected.

  Check the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

Module cni-simple-bridge

- D8IPv4OptionsGuardAuditMode CE S9
  ▾ **The CVE-2026-98432 filter only counts packets on VALUE nodes of the cluster.**

  The `ipv4OptionsGuard.mode` of the CNI module is `Audit` on VALUE nodes: packets with IP options are only counted, not dropped, and MPTCP stays enabled. The nodes remain exposed to CVE-2026-98432.

  Clusters that existed before the mitigation get `Audit` by default. Once the `D8IPv4OptionsGuardMatchedTraffic` alert shows that no workload relies on IP options, switch to `Enforce` in the ModuleConfig of the CNI module (`cni-cilium`, `cni-flannel` or `cni-simple-bridge`):

```
spec:
  settings:
    ipv4OptionsGuard:
      mode: Enforce
```

- D8IPv4OptionsGuardBackendMismatch CE S6
  ▾ The CVE-2026-98432 filter on the node NODE_NAME uses a different iptables mode than iptables-wrapper selected.

  The node `NODE_NAME` is protected, but the filter uses `XXX`: the kernel modules of the mode selected by iptables-wrapper are not loaded, while those of the other mode are.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The filter and the other Deckhouse components use different packet filtering frameworks on this node. Such a security configuration is harder to audit and troubleshoot.

  Possible actions:

  - Find out why the kernel modules of the mode selected by iptables-wrapper are not loaded on the node.
  - If loading these kernel modules is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.
- D8IPv4OptionsGuardFilterBypassed CE S6
  ▾ Packets can skip the CVE-2026-98432 filter in VALUE network namespaces on the node NODE_NAME.

  The node `NODE_NAME` uses legacy iptables. In VALUE network namespaces, a rule in the `PREROUTING` chain of the `raw` table ahead of the jump to `D8-IPV4OPT-GUARD` can end the traversal of the table (for example, `-j ACCEPT`). Packets accepted by that rule never reach the filter.

  The guard puts its jump first when it installs the filter, but does not move it later, so as not to fight with other software on the node that manages its own rules. The affected network namespaces are named in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  Possible actions:

  - Find the software that inserted the rule (`iptables-legacy -t raw -S PREROUTING` in the network namespace) and make it append the rule instead, or use a target that does not end the traversal.
  - Or put the jump first by hand: `iptables-legacy -t raw -I PREROUTING 1 -j D8-IPV4OPT-GUARD`. The guard removes the old jump on its next pass.
- D8IPv4OptionsGuardMatchedTraffic CE S7
  ▾ IPv4 packets with IP options were seen on the node NODE_NAME in the Audit mode.

  Over the last hour, the ipv4-options-guard sidecar matched VALUE IPv4 packets with IP options (rule `XXX`) on the node `NODE_NAME`.

  In the `Audit` mode they are only counted. In the `Enforce` mode they would be dropped, and this alert is not raised.

  Legitimate TCP traffic almost never carries IP options. If a workload relies on them (for example, CIPSO labeled networking), exclude the node with the `ipv4-options-guard.deckhouse.io/disabled: "true"` label before switching `ipv4OptionsGuard.mode` to `Enforce`. Otherwise, this may be an attempt to exploit CVE-2026-98432.

- D8IPv4OptionsGuardMetricsAbsent CE S4
  ▾ The CVE-2026-98432 filter state is unknown on the node NODE_NAME.

  The CNI agent pod runs on the node `NODE_NAME`, but Prometheus gets no metrics from its `ipv4-options-guard` container.

  Without these metrics, the other `D8IPv4OptionsGuard*` alerts cannot fire for this node, so it may be unprotected unnoticed.

  Check the `ipv4-options-guard` container of the CNI agent pod on that node and the scraping of its metrics.

- D8IPv4OptionsGuardNodeUnprotected CE S4
  ▾ The CVE-2026-98432 filter cannot be installed on the node NODE_NAME.

  The ipv4-options-guard sidecar of the CNI agent could not protect the node `NODE_NAME` (reason: `XXX`).

  - `iptables_wrapper_failed` — iptables-wrapper could not determine the iptables mode of the node.
  - `kernel_modules_not_loaded` — the kernel modules of the mode selected by iptables-wrapper are not loaded, and loading them is not allowed.

  The details are in the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

  The node remains exposed to CVE-2026-98432 until its kernel is updated.

  Possible actions:

  - Update the node kernel to a version with the fix and reboot the node.
  - If the node uses legacy iptables and loading the kernel modules named in the reason is acceptable, set `ipv4OptionsGuard.allowKernelModuleLoad: true` in the CNI module settings.
- D8IPv4OptionsGuardReconcileFailed CE S6
  ▾ **The CVE-2026-98432 filter failed in VALUE network namespaces on the node NODE_NAME.**

  Some network namespaces on the node `NODE_NAME` are not protected.

  Check the logs of the `ipv4-options-guard` container of the CNI agent pod on that node.

Module control-plane-manager

- D8ControlPlaneComponentsResourcesInsufficientCapacity CE S5
  ▾ **Automatic control-plane resource tuning cannot increase XXX requests due to lack of available capacity.**

  Automatic control-plane resource tuning attempted to increase `XXX` requests for one or more control-plane components (kube-apiserver, etcd, kube-controller-manager, kube-scheduler), but the calculated requests exceed the available capacity of at least one master node after accounting for the kubelet reservation and resource requests of other (non-control-plane) pods already scheduled on the node.

  XXX Additional memory required: `XXXB`. XXX Additional CPU required: `VALUEm` CPU. XXX

  To resolve this, free up resources on master nodes by moving or reducing other workloads or increase the CPU or RAM available on the nodes.

  To disable automatic tuning for this resource and manage requests manually, set `spec.settings.resourcesRequests.XXX` in the ModuleConfig of `control-plane-manager`.

- D8ControlPlaneDefaultVersionDrift CE S5
  ▾ **Control plane Kubernetes version rollback is blocked (Default soft-guard freeze).**

  The cluster tracks the Deckhouse default Kubernetes version (`kubernetesVersion: Default` in ModuleConfig `control-plane-manager`, or unset ModuleConfig with no ClusterConfiguration pin). That default is now more than 1 minor below the highest version this cluster has ever run (`maxUsed`).

  Rollback blocked: Deckhouse freezes the published target (holds it at the current control-plane version, or at the last desired version when the current one is not published yet) and does not roll Default down toward the lower default. Control plane and kubelet stay on the frozen target until the Deckhouse default catches up within the `maxUsed−1` window.

  Check the `frozen` label on the metric: `frozen="true"` means the digit is held as described above. `frozen="false"` means the window is violated but nothing was left to hold the digit at (for example the `d8-cluster-kubernetes` ConfigMap was deleted), so the published target **has** moved down to the Deckhouse default — pin the intended version explicitly.

  To resolve this alert:

  - Do nothing and wait, if you are happy staying on the version the cluster runs now. This is a valid resolution only while `frozen="true"`: the cluster is not degraded, the version is held, and the alert clears by itself once the Deckhouse default rises back into the `maxUsed−1` window. How long that takes depends on why the default dropped — after a Deckhouse rollback, until you upgrade Deckhouse again; when the release default is simply older than the version this cluster runs, until a Deckhouse release ships a high enough default, which may be several releases away.
  - To keep the current (higher) version and stop the alert now, pin it explicitly:

```
d8 k patch mc control-plane-manager --type merge -p '{"spec":{"settings":
{"kubernetesVersion":"current version"}}}'
```

  - To downgrade deliberately, pin one minor at a time (more than 1 minor below `maxUsed` is rejected):

```
d8 k patch mc control-plane-manager --type merge -p '{"spec":{"settings":
{"kubernetesVersion":"current version - 1 minor"}}}'
```

- D8ControlPlaneManagerObsoleteMonitoringKubernetesControlPlaneModuleConfig CE S9
  ▼ Obsolete monitoring-kubernetes-control-plane ModuleConfig should be removed.

  The `monitoring-kubernetes-control-plane` module was merged into the `control-plane-manager` module, but its ModuleConfig still exists in the cluster and is no longer used.

  Remove it manually:

```
d8 k delete moduleconfig monitoring-kubernetes-control-plane
```

  If the ModuleConfig is deployed from an external source (e.g. a Git repository), make sure it is removed from there so it is not created again.

- D8ControlPlaneManagerPodNotRunning CE S6
  ▼ Controller pod isn't running on node NODE_NAME.

  The `d8-control-plane-manager` pod is either failing or hasn’t been scheduled on node `NODE_NAME`.

  To resolve this issue, check the status of the `kube-system/d8-control-plane-manager` DaemonSet and its pods by running the following command:

```
d8 k -n kube-system get daemonset,pod --selector=app=d8-control-plane-manager
```

- D8ControlPlaneNodeMaintenanceModeEnabled CE S4
  ▼ Maintenance mode is enabled for control-plane node NODE_NAME.

  Control-plane node `NODE NAME` has maintenance mode enabled with label `control-plane-manager.deckhouse.io/maintenance`. While this label is present, the control plane is not managed by `control-plane-manager`.

  If maintenance is complete, remove the label:

```
d8 k label cpn NODE_NAME control-plane-manager.deckhouse.io/maintenance-
```

- D8ControlPlaneOperationRunningTooLong CE S4
  ▼ Control-plane operation XXX for COMPONENT_NAME on node NODE_NAME runs or retries longer than 10 minutes.

  Control-plane operation `XXX` for component `COMPONENT_NAME` on node `NODE_NAME` has been running or retrying for more than 10 minutes.

  Check the corresponding ControlPlaneOperation and logs of the `d8-control-plane-manager` Pod on that node:

```
d8 k get cpo XXX -o yaml
d8 k -n kube-system logs -l app=d8-control-plane-manager --field-selector
spec.nodeName=NODE_NAME
```

- D8EtcdDatabaseHighFragmentationRatio CE S7
  ▼ etcd database size in use is less than 50% of the allocated storage.

  The etcd database size in use on instance `INSTANCE_NAME` is less than 50% of the allocated disk space, indicating potential fragmentation. Additionally, the total storage size exceeds 75% of the configured quota.

  To resolve this issue, defragment the etcd database by running the following command:

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

- D8EtcdExcessiveDatabaseGrowth CE S4
  ▼ **etcd cluster database is growing rapidly.**

  Based on the growth rate observed over the last six hours, Deckhouse predicts that the etcd database will run out of disk space within one day on instance `INSTANCE_NAME`.

  To prevent disruptions, investigate the cause and take necessary action.

- D8EtcdHighRequestsLatency CE S5
  ▼ **Etcd request latency (p99) has exceeded 800 ms multiple times in the past hour.**

  The 99th percentile of etcd request duration on `INSTANCE_NAME` has exceeded 800 ms more than twice in the last hour.

  High etcd latency can cause API server timeouts, slow admission webhooks, and general cluster instability.

  Investigate the following:

  - **Node disk I/O**: Etcd is extremely sensitive to disk performance. Check disk utilization and I/O wait on the control plane node.
  - **Network-attached storage**: If the etcd data directory resides on NFS, Ceph, or another network storage, check connectivity and latency to the storage backend. Etcd requires a consistently low-latency local disk (SSD recommended).
  - **Control plane summary**: Review the **Kubernetes Cluster → Control Plane Status** Grafana dashboard for an overall picture of API server and etcd health, request rates, and error rates.
  - **Node base performance**: Check node CPU and RAM utilization and load average. If the node is CPU-bound (load average significantly exceeds the number of CPU cores), consider scaling up the control plane node.

- D8InvalidEtcdDefragCronSchedule CE S6
  ▼ **Invalid etcd defragmentation cron schedule in ModuleConfig.**

  The `controlPlaneManager.etcd.defrag.cronSchedule` value `XXX` is not a valid cron expression.

  Deckhouse has fallen back to the default schedule (`0 1 * * *`) to keep defragmentation running.

  To fix this, set a valid 5-field cron expression:

  ```
  d8 k patch moduleconfig control-plane-manager --type merge \
    -p '{"spec":{"settings":{"etcd":{"defrag":{"cronSchedule":"0 1 * * *"}}}}}'
  ```

  Valid examples:

  - `0 1 * * *` — daily at 01:00 UTC
  - `0 */6 * * *` — every 6 hours
  - `30 2 * * 0` — weekly on Sunday at 02:30 UTC

- D8KubeEtcdDatabaseSizeCloseToTheLimit CE S3
  ▼ **etcd database size is approaching the limit.**

  The etcd database size on `NODE_NAME` is nearing its size limit. This may be caused by a high number of events, such as pod evictions or the recent creation of numerous resources in the cluster.

  Possible solutions:

  - Defragment the etcd database by running the following command:

    ```
    d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
      --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt \
      --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
    ```

  - Increase node memory. Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

    Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
…	…

- D8KubernetesStaleTokensDetected CE S8
  ▼ Stale service account tokens detected for XXX/XXX.

  Service account `XXX/XXX` is using a stale token. This issue may occur if an application reads the token only at startup and does not reload it periodically. As a result, an outdated token might be used, leading to security breach and authentication failures.

  Recommended actions:

  - Ensure your application is configured to periodically reload the token from the file system.
  - Verify that you are using an up-to-date client library that supports automatic token rotation.

  Note that currently these tokens are not blocked because the `--service-account-extend-token-expiration` flag is enabled by default (`Default: true`). With this flag enabled, admission-injected tokens are extended up to 1 year during token generation to facilitate a safe transition from legacy tokens to the bound service account token feature, ignoring the value of `service-account-max-token-expiration`.

  For further investigation: You can find the problematic pods using the following command:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.serviceAccountName=="XXX"
and .metadata.namespace=="XXX") | .metadata.name'
```

- D8KubernetesVersionIsDeprecated CE S7
  ▼ Kubernetes version VERSION_NUMBER is deprecated.

  The current Kubernetes version `VERSION_NUMBER` has been deprecated, and support for it will be removed in upcoming releases.

  Please migrate to the next kubernetes version (at least 1.34)

  Read documentation about how to update the Kubernetes version in the cluster.

- D8NeedDecreaseEtcdQuotaBackendBytes CE S6
  ▼ Deckhouse suggests reducing quota-backend-bytes.

  When the control plane node memory is reduced, Deckhouse may suggest reducing `quota-backend-bytes`. While Deckhouse is capable of automatically increasing this value, reducing it must be done manually.

  To modify `quota-backend-bytes`, set the `controlPlaneManager.etcd.maxDbSize` parameter. Before setting a new value, check the current database usage on every control plane node by running:

```
for pod in $(d8 k get pod -n kube-system -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec -ti "$pod" -- /usr/bin/etcdctl \
    --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt \
    --key /etc/kubernetes/pki/etcd/ca.key \
    endpoint status -w json | jq --arg a "$pod" -r \
    '.[0].Status.dbSize / 1024 / 1024 | tostring | $a + ": " + . + " MB"';
done
```

Things to note:

- The maximum value for `controlPlaneManager.etcd.maxDbSize` is 8 GB.
- If control plane nodes have less than 24 GB, set `controlPlaneManager.etcd.maxDbSize` to 2 GB.
- Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

  Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
…	…

- D8ObsoleteClusterDomainInClusterConfiguration CE S6
  ▼ The obsolete clusterDomain field is still present in ClusterConfiguration.

  `ClusterConfiguration` still carries the obsolete `clusterDomain` field. The place to manage the cluster domain is ModuleConfig `control-plane-manager`.

  This fires simply because the field is present, even if ModuleConfig `control-plane-manager` already overrides it - the goal is to remove the deprecated field, not just override it.

  To resolve this alert:

  1. If ModuleConfig `control-plane-manager` does not exist, create it:

     ```
     d8 s module enable control-plane-manager
     ```

  2. Migrate the value straight from the Secret into ModuleConfig:

     ```
     DOMAIN=$(d8 k -n kube-system get secret d8-cluster-configuration \
       -o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d \
       | sed -n 's/^clusterDomain:[[:space:]]*//p' | tr -d "\"'")

     d8 k patch moduleconfig control-plane-manager --type merge \
       -p "{\"spec\":{\"version\":3,\"settings\":{\"network\":
     {\"clusterDomain\":\"$DOMAIN\"}}}}"
     ```

     The value must match the one the cluster currently runs with; admission rejects a mismatch.

  3. Remove the field from ClusterConfiguration:

     ```
     d8 system edit cluster-configuration
     ```

     Delete `clusterDomain`, then save.

  Once the field is removed from `ClusterConfiguration`, this alert is resolved.

- D8ObsoleteEncryptionAlgorithmInClusterConfiguration CE S6
  ▼ Obsolete encryptionAlgorithm in ClusterConfiguration needs to be migrated to control-plane-manager ModuleConfig.

  The `encryptionAlgorithm` field is set to a non-default value in ClusterConfiguration but has not been migrated to ModuleConfig `control-plane-manager` yet.

  Until the migration is complete, the value from ClusterConfiguration is used to avoid unintended

certificate rotation.

To resolve this alert, migrate the setting:

1. If ModuleConfig `control-plane-manager` does not exist, create it first:

   ```
   d8 s module enable control-plane-manager
   ```

2. Migrate the value from ClusterConfiguration to ModuleConfig in one command:

   ```
   d8 k patch moduleconfig control-plane-manager --type merge -p "$(cat <<EOF
   {"spec": {"version": 3, "settings": {"encryptionAlgorithm": "$(d8 k -n kube-system get secret d8-cluster-configuration -o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d | grep encryptionAlgorithm | awk '{print $2}')"}}}
   EOF
   )"
   ```

Once ModuleConfig is updated, this alert will be resolved automatically.

1. Remove `encryptionAlgorithm` from ClusterConfiguration:

   ```
   d8 system edit cluster-configuration
   ```

   Delete the `encryptionAlgorithm` field and save.

- D8ObsoleteGlobalControlPlaneResourcesRequests CE S6
  ▼ **Obsolete global control-plane resource requests need to be migrated.**

  Control-plane resource requests are being migrated from global settings to `moduleconfig control-plane-manager`.

  The obsolete settings are still used as a fallback, but they should be moved to the `control-plane-manager` module settings. The migration does not change the meaning of the values: `cpu` and `memory` remain the total resource requests for control-plane components on each master node.

  To complete the migration, follow the steps below:

  1. Review the current obsolete settings in `moduleconfig global`:

     ```
     d8 k get mc global -o jsonpath='{.spec.settings.modules.resourcesRequests.controlPlane}'; echo
     ```

     Example of obsolete settings:

     ```
     spec:
       settings:
         modules:
           resourcesRequests:
             controlPlane:
               cpu: 1000m
               memory: 500Mi
     ```

  2. Make sure the `moduleconfig control-plane-manager` resource exists and uses settings version `3`.

     If the resource does not exist, create it:

     ```
     d8 s module enable control-plane-manager
     ```

     Set the settings version:

     ```
     d8 k patch mc control-plane-manager --type merge \
       -p '{"spec":{"version":3}}'
     ```

3. Copy the obsolete values from `moduleconfig global` to `moduleconfig control-plane-manager`.

   The following command preserves the original JSON value types, including numeric CPU values:

   ```
   d8 k patch mc control-plane-manager --type merge -p "$(
     d8 k get mc global -o json | jq -c \
       '{"spec":{"version":3,"settings":{"resourcesRequests":
   .spec.settings.modules.resourcesRequests.controlPlane}}}'
   )"
   ```

   After migration, the `moduleconfig control-plane-manager` resource should contain:

   ```
   spec:
     version: 3
     settings:
       resourcesRequests:
         cpu: 1000m
         memory: 500Mi
   ```

4. Check that the values were applied:

   ```
   d8 k get mc control-plane-manager -o
   jsonpath='{.spec.settings.resourcesRequests}'; echo
   ```

5. Remove the obsolete settings from `moduleconfig global`:

   ```
   d8 k patch mc global --type merge \
     -p '{"spec":{"settings":{"modules":{"resourcesRequests":null}}}}'
   ```

- D8ObsoleteKubernetesVersionFieldInClusterConfiguration CE S6
  ▾ The obsolete kubernetesVersion field is still present in ClusterConfiguration.

  `ClusterConfiguration` still carries the obsolete `kubernetesVersion` field. The place to manage the cluster Kubernetes version is `settings.kubernetesVersion` in ModuleConfig `control-plane-manager`.

  To resolve this alert:

  1. If ModuleConfig `control-plane-manager` does not exist, create it:

     ```
     d8 s module enable control-plane-manager
     ```

  2. Read what the obsolete field currently holds:

     ```
     d8 k -n kube-system get secret d8-cluster-configuration -o jsonpath='{.data.cluster-
     configuration\.yaml}' | base64 -d | grep '^kubernetesVersion:'
     ```

  3. If ModuleConfig has no `kubernetesVersion` yet, set that same value there, so the cluster keeps the version it runs today (replace `1.34` with what you saw; if you saw `Automatic`, use `Default`):

     ```
     d8 k patch moduleconfig control-plane-manager --type merge -p '{"spec":
     {"version":3,"settings":{"kubernetesVersion":"1.34"}}}'
     ```

     Set `Default` instead if you want Deckhouse to choose the version. Note that this starts an upgrade whenever the release default is higher than the pinned version:

     ```
     d8 k patch moduleconfig control-plane-manager --type merge -p '{"spec":
     {"version":3,"settings":{"kubernetesVersion":"Default"}}}'
     ```

  4. Remove `kubernetesVersion` from ClusterConfiguration:

```
d8 system edit cluster-configuration
```

Delete the `kubernetesVersion` field and save.

Once the `kubernetesVersion` field is removed from `ClusterConfiguration`, this alert is resolved.

- D8ObsoleteNetworkFieldsInClusterConfiguration CE S6
  ▼ **Obsolete network fields are still present in ClusterConfiguration.**

  `ClusterConfiguration` still carries one or more of the obsolete `podSubnetCIDR`, `serviceSubnetCIDR`, `podSubnetNodeCIDRPrefix` fields. The place to manage the cluster's network is the `network` section of ModuleConfig `control-plane-manager`.

  This fires simply because a field is present, even if ModuleConfig `control-plane-manager` already overrides it - the goal is to remove the deprecated field, not just override it.

  To resolve this alert:

  1. If ModuleConfig `control-plane-manager` does not exist, create it:

     ```
     d8 s module enable control-plane-manager
     ```

  2. Migrate the values straight from the Secret into ModuleConfig using the following command:

     ```
     d8 k patch moduleconfig control-plane-manager --type merge -p "$(
       d8 k -n kube-system get secret d8-cluster-configuration -o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d \
         | yq -o=json '{"podSubnetCIDR": .podSubnetCIDR, "serviceSubnetCIDR": .serviceSubnetCIDR, "podSubnetNodeCIDRPrefix": .podSubnetNodeCIDRPrefix}' \
         | jq -c '{spec: {version: 3, settings: {network: with_entries(select(.value != null))}}}'
     )"
     ```

     Once a field is set in ModuleConfig this way, it cannot be migrated again: the network group is immutable once set.

  3. Remove all three fields from ClusterConfiguration:

     ```
     d8 system edit cluster-configuration
     ```

     Delete `podSubnetCIDR`, `serviceSubnetCIDR` and `podSubnetNodeCIDRPrefix`, then save.

  Once all three fields are removed from `ClusterConfiguration`, this alert is resolved.

- D8ObsoletePublishAPIinUserAuthn CE S6
  ▼ **Obsolete Publish API settings in user-authn need to be migrated.**

  Publish API ingress settings are being migrated from `mc user-authn` to `mc control-plane-manager`.

  To complete the migration, follow the steps below:

  1. Review current settings for `publishAPI` in the `moduleconfig user-authn` resource.

     Note that legacy `user-authn` `spec.version: 1` uses `publishAPI.enable` instead of `publishAPI.enabled`. No manual rewrite is required: the command in step 4 uses the normalized payload from the temporary ConfigMap.

  2. If the `moduleconfig control-plane-manager` resource exists, update its spec to `spec.version: 3`. If `settings.apiserver.loadBalancer` is set, migrate it to `settings.apiserver.publishAPI.loadBalancer`. If the resource does not exist, proceed to step 3.

  3. If `moduleconfig control-plane-manager` does not exist, create it via `d8`:

     ```
     d8 s module enable control-plane-manager
     ```

     And set the settings version:

```
d8 k patch moduleconfig control-plane-manager --type merge \
-p '{"spec":{"version":3}}'
```

4. Apply the migrated settings into `moduleconfig control-plane-manager` from temporary configMap by executing the following script:

```
d8 k patch moduleconfig control-plane-manager --type merge -p "$(cat <<EOF
{"spec": {"settings": {"apiserver": {"publishAPI": {"ingress": \
$(d8 k -n kube-system get cm d8-publishapi-config-migration -
ojsonpath='{.data.config}')
   }}}}}
EOF
)"
```

5. Remove obsolete `spec.settings.publishAPI` section from `moduleconfig user-authn`.

- D8ProblematicFeatureGateInUse CE S9
  ▼ Feature gate XXX is STATUS_REFERENCE in current Kubernetes version XXX and should be removed from the configuration. Is not using in the cluster control-plane.

  The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it is STATUS_REFERENCE in the current Kubernetes version `XXX`.

  XXX This feature gate does not exist in any component (apiserver, kube-controller-manager, kube-scheduler, kubelet) and is not being applied to the cluster. XXX

  Recommended actions:

  1. Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

```
d8 k edit mc control-plane-manager
```

  For more information about Kubernetes feature gates, refer to the official documentation.

- D8UsingFeatureGateWillBeDeprecated CE S9
  ▼ Feature gate XXX will be deprecated in Kubernetes version XXX.

  The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it will be deprecated in Kubernetes version `XXX`.

  Current Kubernetes version: `XXX`

  Recommended actions:

  1. Review the usage of this feature gate in your cluster.
  2. Plan migration away from this feature gate before upgrading to version `XXX`.
  3. Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

```
d8 k edit mc control-plane-manager
```

  For more information about Kubernetes feature gates, refer to the official documentation.

- HelmReleasesHasResourcesWithDeprecatedVersions CE S5
  ▼ At least one Helm release contains resources with deprecated apiVersion, which will be removed in Kubernetes version VERSION_NUMBER.

  To list all affected resources, run the following Prometheus query:

```
max by (helm_release_namespace, helm_release_name, helm_version,
resource_namespace, resource_name, api_version, kind, k8s_version)
(resource_versions_compatibility) == 1
```

  For more details on upgrading deprecated resources, refer to the Kubernetes deprecation guide available at `https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX`.

  Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- HelmReleasesHasResourcesWithUnsupportedVersions CE S4

▾ At least one Helm release contains resources with unsupported apiVersion for Kubernetes version VERSION_NUMBER.

To list all affected resources, run the following Prometheus query:

```
max by (helm_release_namespace, helm_release_name, helm_version,
resource_namespace, resource_name, api_version, kind, k8s_version)
(resource_versions_compatibility) == 2
```

For more details on migrating deprecated resources, refer to the Kubernetes deprecation guide available at `https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX`.

Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- K8SApiserverDown CE S3
  ▾ API servers can't be reached.

  No API servers are reachable, or they have all disappeared from service discovery.

- K8sCertificateExpiration CE S5
  ▾ Kubernetes has API clients with soon-to-expire certificates.

  Some clients are connecting to COMPONENT_NAME with certificates that will expire in less than a day on node `COMPONENT_NAME`.

  To check control plane certificates, use kubeadm:

  1. Install kubeadm using the following command:

     ```
     apt install kubeadm=1.24.*
     ```

  2. Check certificates:

     ```
     kubeadm alpha certs check-expiration
     ```

  To check kubelet certificates, do the following on each node:

  1. Check kubelet configuration:

     ```
     ps aux \
       | grep "/usr/bin/kubelet" \
       | grep -o -e "--kubeconfig=\S*" \
       | cut -f2 -d"=" \
       | xargs cat
     ```

  2. Locate the `client-certificate` or `client-certificate-data` field.
  3. Check certificate expiration using OpenSSL.

  Note that there are no tools to find other stale kubeconfig files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8sCertificateExpiration CE S6
  ▾ Kubernetes has API clients with soon-to-expire certificates.

  Some clients are connecting to COMPONENT_NAME with certificates that will expire in less than 7 days on node `NODE_NAME`.

  To check control plane certificates, use kubeadm:

  1. Install kubeadm using the following command:

     ```
     apt install kubeadm=1.24.*
     ```

  2. Check certificates:

     ```
     kubeadm alpha certs check-expiration
     ```

  To check kubelet certificates, do the following on each node:

1. Check kubelet configuration:

   ```
   ps aux \
     | grep "/usr/bin/kubelet" \
     | grep -o -e "--kubeconfig=\S*" \
     | cut -f2 -d"=" \
     | xargs cat
   ```

2. Locate the `client-certificate` or `client-certificate-data` field.
3. Check certificate expiration using OpenSSL.

Note that there are no tools to find other stale kubeconfig files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8SControllerManagerTargetDown CE S3
  ▼ **Controller manager is down.**

  There is no running kube-controller-manager. As a result, deployments and replication controllers are not progressing.

- K8SSchedulerTargetDown CE S3
  ▼ **Scheduler is down.**

  The Kubernetes scheduler is not running. As a result, new pods are not being assigned to nodes.

- KubeEtcdHighFsyncDurations CE S7
  ▼ **Syncing (fsync) WAL files to disk is slow.**

  In the last 15 minutes, the 99th percentile of the fsync duration for WAL files exceeded 0.5 seconds: VALUE.

  Possible causes:

  - High disk latency where etcd data is stored.
  - High CPU usage on the node.

- KubeEtcdHighNumberOfLeaderChanges CE S5
  ▼ **The etcd cluster is re-electing the leader too frequently.**

  There have been VALUE leader re-elections for the etcd cluster member running on node `NODE_NAME` in the last 10 minutes.

  Possible causes:

  - High disk latency where etcd data is stored.
  - High CPU usage on the node.
  - Degradation of network connectivity between cluster members in the multi-master mode.

- KubeEtcdInsufficientMembers CE S4
  ▼ **Insufficient members in the etcd cluster.**

  The etcd cluster has too few members, increasing the risk of failure if another member becomes unavailable. To resolve this issue, check the status of etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd`

- KubeEtcdNoLeader CE S4
  ▼ **The etcd cluster member running on node NODE_NAME has lost the leader.**

  To resolve this issue, check the status of the etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd | grep NODE_NAME`

- KubeEtcdTargetAbsent CE S5
  ▼ **There is no etcd target in Prometheus.**

  Steps to troubleshoot:

  1. Check the status of the etcd Pods:

     ```
     d8 k -n kube-system get pod -l component=etcd
     ```

  2. Review Prometheus logs:

     ```
     d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -c prometheus
     ```

- KubeEtcdTargetDown CE S5

▼ **Prometheus is unable to scrape etcd metrics.**

Steps to troubleshoot:

1. Check the status of the etcd Pods:

   ```
   d8 k -n kube-system get pod -l component=etcd
   ```

2. Review Prometheus logs:

   ```
   d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -c prometheus
   ```

- KubernetesServiceExternalIPsDeprecated CE S6
  ▼ **The cluster has Services that use the deprecated spec.externalIPs field.**

  There are `VALUE` Service(s) in user namespaces that use the `spec.externalIPs` field. Services in the `d8-*` and `kube-system` namespaces are not checked.

  The `spec.externalIPs` field is officially deprecated in Kubernetes because of the CVE-2020-8554 security vulnerability. Starting with Kubernetes 1.36, the API server returns a warning for every request that sets this field. Support for the field will be removed in future Kubernetes versions.

  Plan the migration of the affected Services to a supported way of exposing them, for example:

  - A Service of the `LoadBalancer` type. In bare-metal clusters, use the `metallb` module to assign addresses.
  - A Service of the `NodePort` type.
  - An Ingress or Gateway API resource for HTTP(S) traffic.

  For details, refer to the Kubernetes announcement.

  To list the affected Services with their namespaces and external IP addresses, run:

  ```
  d8 k get services -A -o json | jq -r '["NAMESPACE", "NAME", "TYPE", "EXTERNAL-IPS"],
  (.items[] | select(.spec.externalIPs) | select(.metadata.namespace |
  test("^(d8-.*|kube-system)$") | not) | [.metadata.namespace, .metadata.name,
  .spec.type, (.spec.externalIPs | join(","))]) | @tsv' | column -t
  ```

  Alternatively, run the following Prometheus query:

  ```
  max by (namespace, service, external_ip)
  (kube_service_spec_external_ip{namespace!~"d8-.*|kube-system"})
  ```

Module deckhouse

- ApplicationHorizontalPodAutoscalerMissing CE S5
  ▼ **Application NAME in namespace XXX has no HorizontalPodAutoscaler for XXX XXX.**

  The `spec.resourceRequests` parameter of Application `NAME` in namespace `XXX` sets a replicas range (`minReplicas` and `maxReplicas`) for XXX `XXX`, but the application deploys no HorizontalPodAutoscaler for this workload. Deckhouse Platform runs the workload with `minReplicas` replicas instead.

  To fix the issue, do one of the following:

  - Check the application settings: a package can deploy HorizontalPodAutoscaler only when a setting enables it.
  - Replace the range with a fixed number of replicas: set `replicas` instead of `minReplicas` and `maxReplicas`.

  To view the resource settings of the application, run the following command:

  ```
  d8 k get application NAME -n XXX -o jsonpath='{.spec.resourceRequests}'
  ```

- ApplicationHorizontalPodAutoscalerPinned CE S5
  ▼ **HorizontalPodAutoscaler of XXX XXX of Application NAME in namespace XXX doesn't scale the workload.**

  The `spec.resourceRequests` parameter of Application `NAME` in namespace `XXX` sets a fixed number of replicas (`replicas`) for XXX `XXX`, which a HorizontalPodAutoscaler of the application

scales. Deckhouse Platform sets both bounds of the HorizontalPodAutoscaler to this number, so it doesn’t scale the workload.

To let HorizontalPodAutoscaler scale the workload, replace `replicas` with a range: set `minReplicas` and `maxReplicas`.

To view the resource settings of the application, run the following command:

```
d8 k get application NAME -n XXX -o jsonpath='{.spec.resourceRequests}'
```

- ApplicationIsInMaintenanceMode CE S6
  ▼ Application NAME in namespace XXX is running in maintenance mode.

  Application `NAME` in namespace `XXX` is running in maintenance mode. In this mode, its state is not reconciled, which prevents configuration or hook changes from being applied automatically.

  To switch the application back to normal mode, edit the corresponding Application resource with the following command:

  ```
  d8 k patch application NAME -n XXX --type=json -p='[{"op": "remove", "path": "/spec/maintenance"}]'
  ```

- ApplicationVerticalPodAutoscalerMissing CE S5
  ▼ VerticalPodAutoscaler does not manage at least one resource range of container CONTAINER_NAME of XXX XXX of Application NAME in namespace XXX.

  The `spec.resourceRequests` parameter of Application `NAME` in namespace `XXX` sets a resource range (`min` and `max`) for container `CONTAINER_NAME` of `XXX XXX`, but at least one resource in the range is not managed by a VerticalPodAutoscaler of the application. Deckhouse Platform applies the range for each such resource as fixed values instead: the container gets `min` as the request and `max` as the limit.

  To fix the issue, do one of the following:

  - Check that the `vertical-pod-autoscaler` module is enabled: an application deploys VerticalPodAutoscaler only when the module is enabled.
  - Check the VerticalPodAutoscaler of the workload: it doesn’t set the resources in the `Off` update mode, for a container whose policy mode is `Off`, for a resource missing from `controlledResources` of the policy, and for init containers.
  - Replace the range with fixed resources: set `requests` and `limits` instead of `min` and `max`.

  To view the resource settings of the application, run the following command:

  ```
  d8 k get application NAME -n XXX -o jsonpath='{.spec.resourceRequests}'
  ```

- ApplicationVerticalPodAutoscalerPinned CE S5
  ▼ Fixed resource settings limit VerticalPodAutoscaler for container CONTAINER_NAME of XXX XXX of Application NAME in namespace XXX.

  The `spec.resourceRequests` parameter of Application `NAME` in namespace `XXX` sets fixed `cpu` or `memory` requests or limits for container `CONTAINER_NAME` of `XXX XXX`. If a request is set for a resource managed by VerticalPodAutoscaler, Deckhouse Platform sets both VPA bounds to that request. If a limit is set, Deckhouse Platform sets `controlledValues: RequestsOnly`: VPA can change requests that are not fixed by the Application, but does not change limits.

  To let VerticalPodAutoscaler manage the requests within a range, replace the fixed `cpu` and `memory` requests and limits with `min` and `max`.

  To view the resource settings of the application, run the following command:

  ```
  d8 k get application NAME -n XXX -o jsonpath='{.spec.resourceRequests}'
  ```

- D8CNIEnabledMoreThanOne CE S2
  ▼ More than one CNI is enabled in the cluster.

  Deckhouse has detected that multiple CNIs are enabled in the cluster. For the cluster to work correctly, exactly one CNI must be enabled.

  To resolve this issue, follow these steps:

  1. List the CNI modules and their enabled state:

```
d8 k get moduleconfig -o custom-
columns='NAME:.metadata.name,ENABLED:.spec.enabled' | grep -E 'NAME|cni-'
```

2. Decide which CNI the cluster must keep. Check which one actually serves the nodes before disabling anything:

```
d8 k -n d8-cni-cilium get pods 2>/dev/null; d8 k -n d8-cni-flannel get pods 2>/dev/null; d8
k -n d8-cni-simple-bridge get pods 2>/dev/null
```

3. Disable every other CNI module, one at a time:

```
d8 k patch moduleconfig CNI_MODULE_NAME --type=merge -p '{"spec":{"enabled":false}}'
```

Changing the CNI of a running cluster disrupts pod networking. If you are unsure which CNI to keep, contact support before disabling anything.

- D8CNIMisconfigured CE S5
  ▼ The parameters specified in ModuleConfig of CNI_NAME do not match the ones that are actually being used in the cluster.

  This happened because there were several sources for configuring CNI parameters in the cluster, and ModuleConfig did not have the highest priority previously.

  To resolve this issue, the following steps should be taken:

  1. Find the ConfigMap `d8-system/desired-cni-moduleconfig` in the cluster, which contains the actual ModuleConfig settings.

     ```
     d8 k -n d8-system get configmap desired-cni-moduleconfig -o yaml
     ```

  2. Apply this prepared ModuleConfig to the cluster. This will not cause any actual reconfigurations in the cluster and is completely safe.
  3. Once the parameters in “ModuleConfig” match those used in the cluster:
     - this alert will be resolved,
     - and the priority of the sources for configuring CNI will change, and ModuleConfig will become the main source of truth.
  4. After that, you can add the desired parameters to the ModuleConfig CNI and they will be applied immediately.
- D8DeckhouseConfigInvalid CE S5
  ▼ Deckhouse rejected the cluster configuration: one of the ModuleConfig resources is invalid.

  Deckhouse has detected errors in the cluster configuration. The usual causes are `spec.settings` that do not match the module’s settings schema, and a failed conversion of `spec.settings` to the latest schema version.

  This alert does not name the module, because the underlying metric has no labels. Deckhouse puts the exact error into the `status.message` field of the ModuleConfig that caused it, prefixed with `Error:`.

  To resolve this issue, follow these steps:

  1. Find the affected configuration. The `MESSAGE` column holds the error and the `VERSION` column holds the settings schema version currently in use:

     ```
     d8 k get moduleconfig
     ```

  2. Read the full error for the ModuleConfig that reports one (`global` is a valid value for `MODULE_NAME`):

     ```
     d8 k get moduleconfig MODULE_NAME -o jsonpath='{.status.message}{"\n"}'
     ```

  3. Fix `spec.settings`, or `spec.version` if the error mentions a settings version or a conversion:

     ```
     d8 k edit moduleconfig MODULE_NAME
     ```

4. If no ModuleConfig reports an error, check the Deckhouse logs:

   ```
   d8 k -n d8-system logs -f -l app=deckhouse
   ```

- D8DeckhouseCouldNotDeleteModule CE S4
  ▼ **Deckhouse is unable to delete the MODULE_NAME module.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotDiscoverModules CE S4
  ▼ **Deckhouse is unable to discover modules.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunGlobalHook CE S5
  ▼ **Deckhouse is unable to run the HOOK_NAME global hook.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunModule CE S4
  ▼ **Deckhouse is unable to start the MODULE_NAME module.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunModuleHook CE S7
  ▼ **Deckhouse is unable to run the MODULE_NAME/HOOK_NAME module hook.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCustomTargetDown CE S4
  ▼ **Prometheus is unable to scrape custom metrics generated by Deckhouse hooks.**

  Prometheus cannot reach the endpoint that serves metrics exported by Deckhouse hooks. Deckhouse core metrics may still be scraped, so this usually points at the hook metrics endpoint rather than at Deckhouse being down.

  To investigate the issue, follow these steps:

  1. Check that the pod is up:

     ```
     d8 k -n d8-system get pods -l app=deckhouse -o wide
     ```

  2. Look for hook failures in the logs, since a hook that never completes exports no metrics:

     ```
     d8 k -n d8-system logs -l app=deckhouse --tail=100 | grep -i error
     ```

  3. Check that the scrape configuration for hook metrics is still present:

     ```
     d8 k -n d8-monitoring get podmonitor deckhouse -o jsonpath='{range
     .spec.podMetricsEndpoints[*]}{.path}{"\n"}{end}'
     ```

- D8DeckhouseDeprecatedConfigmapManagedByArgoCD CE S4
  ▼ **Deprecated Deckhouse ConfigMap managed by Argo CD.**

  The Deckhouse ConfigMap is no longer used.

  To resolve this issue, remove the `d8-system/deckhouse` ConfigMap from Argo CD.

- D8DeckhouseGlobalHookFailsTooOften CE S9
  ▼ The HOOK_NAME Deckhouse global hook is crashing too frequently.

  The `HOOK_NAME` hook has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseHasNoAccessToRegistry CE S7
  ▼ Deckhouse is unable to connect to the registry.

  Deckhouse can’t connect to the registry to check for new releases. With a release channel set, it checks the channel every 3 minutes; without one, it checks the digest of its own image every 15 seconds. Without access to the registry, automatic updates are unavailable.

  Module images come from the same registry, so pods that need a new image fail to start as well, sometimes with errors that do not mention the registry. The usual causes are expired or changed credentials, a registry certificate that has expired or was replaced with one signed by an unknown authority, and a network or proxy change.

  To investigate the issue, follow these steps:

  1. Read the registry error in the logs. `UNAUTHORIZED` points at the credentials, `x509` at the certificate, and a timeout or a refused connection at the network:

     ```
     d8 k -n d8-system logs -l app=deckhouse --tail=500 | grep -iE 'check deckhouse release|registry'
     ```

  2. Check which registry address the cluster is configured to use:

     ```
     d8 k -n d8-system get secret deckhouse-registry -o jsonpath='{.data.address}' | base64 -d
     ```

  3. If the address is correct, the credentials are the next suspect. Check when the secret was last updated:

     ```
     d8 k -n d8-system get secret deckhouse-registry
     ```

  4. If the error mentions `x509`, check who issued the registry certificate and when it expires. Run the command on a master node and substitute the host from the registry address, and its port if it is not 443:

     ```
     openssl s_client -connect REGISTRY_HOST:443 -servername REGISTRY_HOST </dev/null 2>/dev/null | openssl x509 -noout -issuer -enddate
     ```

  5. Check that the node running Deckhouse can reach the registry at all, for example through a proxy or a firewall rule that changed.

- D8DeckhouseIsHung CE S4
  ▼ Deckhouse is down.

  Deckhouse is probably down, since the `deckhouse_live_ticks` metric in Prometheus has stopped increasing. This metric is expected to increment every 10 seconds.

  To investigate the issue, follow these steps:

  1. Check the pod state and its restart count:

     ```
     d8 k -n d8-system get pods -l app=deckhouse -o wide
     ```

  2. Read the logs. A deadlock or a hook that never returns stops the ticks while the pod still looks healthy:

     ```
     d8 k -n d8-system logs -l app=deckhouse --tail=200
     ```

  3. Check whether a queue stopped moving at the same time:

```
d8 system queue list
```

4. If the pod is alive but stuck, restart it:

```
d8 k -n d8-system rollout restart deploy/deckhouse
```

- D8DeckhouseIsNotOnReleaseChannel CE S9
  ▼ **Deckhouse isn't subscribed to any regular release channels.**

  Deckhouse in this cluster isn’t subscribed to any of the regular release channels: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, or `RockSolid`.

  To resolve this issue, follow these steps:

  1. Check the current release channel used in the cluster:

```
d8 k -n d8-system  get deploy deckhouse -o json | jq
'.spec.template.spec.containers[0].image' -r
```

  2. Subscribe to one of the regular release channels by adjusting the `deckhouse` module configuration.

- D8DeckhouseModuleHookFailsTooOften CE S9
  ▼ **The MODULE_NAME/HOOK_NAME Deckhouse hook is crashing too frequently.**

  The `HOOK NAME` hook of the `MODULE_NAME` module has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

  To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseModuleImageDownloadInterrupted CE S4
  ▼ **Download of the MODULE_NAME module image was interrupted.**

  Deckhouse Platform was downloading the image of the `MODULE_NAME` module, version `VERSION_NUMBER`, from the container registry to build its erofs image, and the download was interrupted. The module version isn’t put on the node running Deckhouse Platform until a download completes.

  The download is interrupted when it doesn’t finish within the registry request timeout, 2 minutes by default, or when the connection to the container registry drops. A slow connection to the container registry is the usual cause.

  To investigate the issue, follow these steps:

  1. Find the reason in the logs. A timeout reads as `context deadline exceeded`, a dropped connection as `unexpected EOF` or `connection reset by peer`:

```
d8 k -n d8-system logs -l app=deckhouse --tail=1000 | grep 'download image layers'
```

  2. Find the ModuleSource the module is downloaded from and the address of its container registry:

```
d8 k get module MODULE_NAME -o jsonpath='{.properties.source}'
d8 k get modulesource SOURCE_NAME -o jsonpath='{.spec.registry.repo}'
```

  3. Check that the image of version `VERSION_NUMBER` is in the container registry:

```
d8 cr ls REGISTRY_ADDRESS/MODULE_NAME
```

     For a private registry, pass the credentials from the ModuleSource:

```
d8 cr ls -u USERNAME -p PASSWORD REGISTRY_ADDRESS/MODULE_NAME
```

  4. Check that the node running Deckhouse Platform reaches the container registry and downloads from it fast enough, for example, through a proxy or a firewall rule that changed.

- D8DeckhouseModuleUpdatePolicyNotFound CE S5
  ▼ **Module update policy not found for MODULE_RELEASE.**

  The module update policy for MODULE_RELEASE is missing.

  To resolve this issue, remove the label from the module release using the following command:

  ```
  d8 k label mr MODULE_RELEASE modules.deckhouse.io/update-policy-
  ```

  A new suitable policy will be detected automatically.

- D8DeckhouseModuleValidationError CE S5
  ▼ **Module configuration failed for module MODULE_NAME.**

  Initial config for module MODULE_NAME is not valid.

  You can get more details via

  ```
  d8 k get mr -l module=MODULE_NAME
  ```

  Provided error: XXX

- D8DeckhouseObsoleteMonitoringDeckhouseModuleConfig CE S9
  ▼ **Obsolete monitoring-deckhouse ModuleConfig should be removed.**

  The `monitoring-deckhouse` module was merged into the `deckhouse` module, but its ModuleConfig still exists in the cluster and is no longer used.

  Remove it manually:

  ```
  d8 k delete moduleconfig monitoring-deckhouse
  ```

  If the ModuleConfig is deployed from an external source (e.g. a Git repository), make sure it is removed from there so it is not created again.

- D8DeckhousePodIsNotReady CE S4
  ▼ **The Deckhouse Pod is NOT Ready.**

  The Deckhouse pod is running but has not become Ready. While this lasts, Deckhouse is not converging the cluster: module updates and configuration changes are not applied.

  To investigate the issue, follow these steps:

  1. Check the pod state and its restart count:

     ```
     d8 k -n d8-system get pods -l app=deckhouse -o wide
     ```

  2. Read the events and probe failures at the bottom of the pod description:

     ```
     d8 k -n d8-system describe pod -l app=deckhouse
     ```

  3. Check the logs. A readiness probe that never succeeds is most often a startup hook that fails or hangs:

     ```
     d8 k -n d8-system logs -l app=deckhouse --tail=200
     ```

  4. Check whether a queue is stuck, which keeps Deckhouse from reporting Ready:

     ```
     d8 system queue list
     ```

- D8DeckhousePodIsNotRunning CE S4
  ▼ **The Deckhouse Pod is NOT Running.**

  There is no running Deckhouse pod in the `d8-system` namespace. Nothing is reconciling the cluster: no module updates, no configuration changes, and no Deckhouse releases are applied.

  To investigate the issue, follow these steps:

  1. Check the pod state. `Pending` points at scheduling, `ImagePullBackOff` at the registry, and `CrashLoopBackOff` at Deckhouse failing to start:

```
d8 k -n d8-system get pods -l app=deckhouse -o wide
```

2. Read the events to find out why the pod is not running:

```
d8 k -n d8-system describe pod -l app=deckhouse
```

3. If the pod is restarting, read the logs of the previous run:

```
d8 k -n d8-system logs -l app=deckhouse --previous --tail=200
```

4. Check that the Deployment itself is present and has the replicas it expects:

```
d8 k -n d8-system get deploy deckhouse
```

- D8DeckhousePodIsRestartingTooOften CE S9
  ▼ Excessive Deckhouse restarts detected.

  Number of restarts in the last hour: VALUE.

  Excessive Deckhouse restarts indicate a potential issue. Normally, Deckhouse should be up and running continuously.

  To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseQueueIsHung CE S6
  ▼ Module queue MODULE_NAME (type XXX, hook HOOK_NAME) is hung.

  The module queue head has remained unchanged for more than 10 minutes.

  Affected queue:

  - Module: `MODULE_NAME`
  - Task type: `XXX`
  - Hook: `HOOK_NAME`

  To investigate the issue, check the logs of the `deckhouse` controller pods:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseQueueIsHungCritical CE S4
  ▼ Critical module queue MODULE_NAME (type XXX, hook HOOK_NAME) is hung.

  The critical module queue head has remained unchanged for more than 10 minutes.

  Affected queue:

  - Module: `MODULE_NAME`
  - Task type: `XXX`
  - Hook: `HOOK_NAME`

  To investigate the issue, check the logs of the `deckhouse` controller pods:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseQueueIsHungGlobal CE S4
  ▼ Global/Parallel task XXX (hook HOOK_NAME) is hung.

  The global/parallel task head has remained unchanged for more than 10 minutes.

  Affected task:

  - Task type: `XXX`
  - Hook: `HOOK_NAME`

  To investigate the issue, check the logs of the `deckhouse` controller pods:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseRegistryModeUnsupported CE S9
  ▼ **The deckhouse ModuleConfig stores the unsupported registry mode XXX.**

  The `deckhouse` ModuleConfig has `spec.settings.registry.mode: XXX`. `Proxy` and `Local` are modes of the v1 implementation of the `registry` module, and they have no effect in Deckhouse Platform 1.78: the registry is managed by the `registry` module and configured in the `registry` ModuleConfig. These values are removed in Deckhouse Platform 1.79, and a `deckhouse` ModuleConfig that still stores one will not pass validation there.

  To resolve this issue, do the following before updating to Deckhouse Platform 1.79:

  1. Check the stored value:

     ```
     d8 k get moduleconfig deckhouse -o jsonpath='{.spec.settings.registry}'
     ```

  2. Remove the `spec.settings.registry` section:

     ```
     d8 k patch moduleconfig deckhouse --type json -p '[{"op": "remove", "path":
     "/spec/settings/registry"}]'
     ```

     Alternatively, set the mode to `Unmanaged` and drop the `proxy` section, which is allowed only in `Proxy` mode:

     ```
     d8 k patch moduleconfig deckhouse --type merge -p '{"spec": {"settings":
     {"registry": {"mode": "Unmanaged", "proxy": null}}}}'
     ```

     Both edits are accepted. Switching the mode to `Proxy` or `Local` is refused.

  3. If the cluster needs what the stored mode was set for, an in-cluster registry or cache, configure it in the `registry` ModuleConfig: `mode: Managed` with `primary.upstream`, and `storage.cache` for an in-cluster cache. Removing the stored mode does not change where the cluster pulls images from, so this step is not required to resolve the alert.

  If the `deckhouse` ModuleConfig is deployed from an external source (e.g. a Git repository), make the same change there so the value is not applied again.

- D8DeckhouseSelfTargetAbsent CE S4
  ▼ **There is no Deckhouse target in Prometheus.**

  Prometheus has no `deckhouse` target at all. Unlike a target that is down, this means the target was never discovered, so the problem is usually with the Service, its endpoints, or the monitoring configuration rather than with Deckhouse itself.

  To investigate the issue, follow these steps:

  1. Check that the Deckhouse pod exists and is running:

     ```
     d8 k -n d8-system get pods -l app=deckhouse -o wide
     ```

  2. Check that the Service has endpoints. An empty `ENDPOINTS` column means nothing matches the selector:

     ```
     d8 k -n d8-system get endpoints deckhouse
     ```

  3. Check that the scrape configuration for Deckhouse exists:

     ```
     d8 k -n d8-system get servicemonitor,podmonitor -l app=deckhouse
     ```

  4. Check that the monitoring stack itself is healthy:

     ```
     d8 k -n d8-monitoring get pods
     ```

- D8DeckhouseSelfTargetDown CE S4
  ▼ **Prometheus is unable to scrape Deckhouse metrics.**

  Prometheus cannot reach the `deckhouse` metrics endpoint. Deckhouse itself may still be working, but its own metrics are missing, so most other Deckhouse alerts are blind while this lasts.

To investigate the issue, follow these steps:

1. Check that the pod is up and how long it has been running:

   ```
   d8 k -n d8-system get pods -l app=deckhouse -o wide
   ```

2. Look for scrape errors, such as a connection refused or a TLS failure:

   ```
   d8 k -n d8-system logs -l app=deckhouse --tail=100
   ```

3. Check that the pod still exposes the `self` metrics port:

   ```
   d8 k -n d8-system get pod -l app=deckhouse -o jsonpath='{range
   .items[*].spec.containers[*].ports[*]}{.name}{"="}{.containerPort}{"\n"}
   {end}'
   ```

- D8DeckhouseWatchErrorOccurred CE S5
  ▼ **Possible API server connection error in the client-go informer.**

  Deckhouse has detected an error in the client-go informer, possibly due to connection issues with the API server.

  Steps to investigate:

  1. Check Deckhouse logs for more information by running:

     ```
     d8 k -n d8-system logs deploy/deckhouse | grep error | grep -i watch
     ```

  2. This alert attempts to detect a correlation between the faulty snapshot invalidation and API server connection errors, specifically for the `handle-node-template` hook in the `node-manager` module.

     To compare the snapshot with the actual node objects for this hook, run the following command:

     ```
     diff -u <(d8 k get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}
     {end}'|sort) <(d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-
     controller module snapshots node-manager -o json | jq '."040-node-
     manager/hooks/handle_node_templates.go"' | jq '.nodes.snapshot[] |
     .filterResult.Name' -r | sort)
     ```

- D8HasModuleConfigAllowedToDisable CE S4
  ▼ **The ModuleConfig annotation for allowing module disabling is set.**

  The ModuleConfig is pending module disabling.

  It is recommended that you keep your module configurations clean by removing unnecessary approval annotations.

  If you ignore this alert and do not clear the annotation, the module may be accidentally removed from the cluster, potentially leading to irreversible consequences.

  To resolve this issue and stop the alert, run the following command:

  ```
  d8 k annotate moduleconfig MODULE_NAME modules.deckhouse.io/allow-disabling-
  ```

  With Module v2 enabled, remove the `packages.deckhouse.io/allow-disabling` annotation instead:

  ```
  d8 k annotate moduleconfig MODULE_NAME packages.deckhouse.io/allow-disabling-
  ```

- D8LicenseClusterKeyInvalid BE S4
  ▼ **The cluster licensing key is damaged.**

  The Secret `d8-system/cluster-key` does not hold a valid cluster key. Deckhouse Platform keeps computing the license state, but it cannot sign the cluster data file for the customer portal, and license records bound to the cluster key are not accepted.

  Deckhouse Platform does not replace a damaged key on its own: a new key is a new cluster

identity, and the license keys issued for the old one may stop applying. Apart from Deckhouse Platform, only `system:sudouser` may change the Secret.

If a backup of the Secret is available, restore it as described in the documentation. Otherwise delete the Secret, and Deckhouse Platform creates a new cluster key:

```
d8 k --as=system:sudouser -n d8-system delete secret cluster-key
```

After a new key is created:

1. Download the cluster data in the Deckhouse web UI section “Licensing”.
2. Upload the file to the customer portal (https://lk.deckhouse.ru) and issue a new key there.
3. Upload the key in the same Deckhouse web UI section.

- D8LicenseComplianceGrace BE S5
  ▼ **The cluster license is in the grace period.**

  The license keys of the cluster have expired, and the grace period is running. When it ends, the license state becomes `Violation`.

  To check the number of remaining grace days in the period, run the following command:

  ```
  d8 k get effectivelicense deckhouse-platform -o jsonpath='{.status.key.graceDays}{"\n"}'
  ```

  To renew the license:

  1. Download the cluster data in the Deckhouse web UI section “Licensing”.
  2. Upload the file to the customer portal (https://lk.deckhouse.ru) and issue a new key there.
  3. Upload the key in the same Deckhouse web UI section.

- D8LicenseComplianceViolation BE S4
  ▼ **The cluster license is violated.**

  The cluster license has been violated: the license keys expired, and the grace period ended (if the key had one).

  Uncovered nodes are reported by the `D8LicenseUnlicensedNodes` alert.

  Deckhouse Platform and the workloads keep running.

  To restore the license, get a new license key:

  1. Download the cluster data in the Deckhouse web UI section “Licensing”.
  2. Upload the file to the customer portal (https://lk.deckhouse.ru) and issue a new key there.
  3. Upload the key in the same Deckhouse web UI section.

- D8LicenseKeyExpiresSoon BE S7
  ▼ **The cluster license expires soon.**

  The license key of the cluster expires within 7 days.

  To renew the license:

  1. Download the cluster data in the Deckhouse web UI section “Licensing”.
  2. Upload the file to the customer portal (https://lk.deckhouse.ru) and issue a new key there.
  3. Upload the key in the same Deckhouse web UI section.

  To check the expiry date of a key, run the following command:

  ```
  d8 k get effectivelicense deckhouse-platform -o jsonpath='{.status.key.validUntil}{"\n"}'
  ```

- D8LicenseRejectedRecords BE S6
  ▼ **There are rejected license records or keys.**

  A record of an installed license key, or a whole key, was rejected. It grants nothing until it is fixed. Possible causes:

  - The key has been damaged, it is not a Deckhouse Platform license key or was not signed by Deckhouse Platform.
  - The key requires a newer Deckhouse Platform version.
  - The key was issued for another cluster.
  - The same record has already been installed with another key.
  - The record does not match the key format.

  Replaced, expired and future records, and records of a type this Deckhouse Platform version does not know, are counted separately and do not fire the alert.

The Deckhouse web UI section “Licensing” shows the cause next to each key in the “License keys” table. To obtain the cause via the command line, run the following command:

```
d8 k get clusterlicenses -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{range
.status.records[?(@.reason)]}  {.id}: {.reason} {.message}{"\n"}{end}{end}'
```

To replace the key, issue a new one in the customer portal (https://lk.deckhouse.ru).

- D8LicenseUnlicensedNodes BE S4
  ▼ **The cluster license is violated because of unlicensed nodes.**

  The cluster has nodes that the installed license keys do not cover, and the license state is `Violation`.

  The state changes only after nodes have stayed uncovered for an hour without a break. During a node replacement the new node joins before the old one leaves, and for that short time one node is uncovered. The delay keeps such a replacement silent.

  Removing the extra nodes ends the violation at once. If nodes go over the limit again within 24 hours after that, the violation ends only after every node has stayed covered for 8 hours, and after 24 hours for every further repeat. While every node is covered and a repeat still goes on, `status.overLimitEndsAt` names the moment it ends.

  To identify the moment the over-limit episode began:

  ```
  d8 k get effectivelicense deckhouse-platform -o jsonpath='{.status.overLimitSince}{"\n"}'
  ```

  The Deckhouse web UI section “Licensing” lists the uncovered nodes in the “License required” group and shows what to add to the license.

  To cover the nodes, get a license key for the current cluster size:

  1. Download the cluster data in the Deckhouse web UI section “Licensing”.
  2. Upload the file to the customer portal (https://lk.deckhouse.ru) and issue a new key there.
  3. Upload the key in the same Deckhouse web UI section.
- D8ModuleOutdatedByMajorVersion CE S9
  ▼ **Module MODULE_NAME is running XXX major versions behind.**

  Module `MODULE_NAME` is running XXX major versions behind on major version releases. Major version updates may contain breaking changes and require immediate attention and careful planning.

  To check the module releases status, run the following command:

  ```
  d8 k get mr -l module=MODULE_NAME
  ```XXXXXX

  This major release has from-to version constraints that must be met before applying the update.
  To check the update instructions (from-to versions), run the following command:

  ```bash
  d8 k get mr NAME -o jsonpath='{.spec.update.versions[*]}'
  ```

  This will show you the update paths including the from-to versions defined in the release.XXX

  To approve the module major release update, run the following command:

  `bash d8 k annotate mr NAME modules.deckhouse.io/approved="true"` XXX
- D8NodeHasDeprecatedOSVersion CE S4
  ▼ **Nodes with deprecated OS versions detected.**

  Deckhouse has detected nodes running deprecated OS versions.

  Steps to troubleshoot:

  1. Get a list of affected nodes by running the following Prometheus query:

     ```
     kube_node_info{os_image=~"Ubuntu 18.04.*|Debian GNU/Linux 10.*|CentOS Linux
     7.*"}
     ```

2. Update the affected nodes to a supported OS version.

- DeckhouseEditionNotFound CE S6
  ▼ **Incorrect Deckhouse Platform edition.**

  Specify a Deckhouse Platform edition in `MODULE_NAME` ModuleConfig.

  ```
  d8 k patch moduleconfig MODULE_NAME --type merge --patch '{"spec": {"settings":
  {"license": {"edition": "DKP_EDITION"}}}}'
  ```

  Where `DKP_EDITION` is the edition you want to use: `CE`, `BE`, `EE`, `SE`, or `SE-plus`.

- DeckhouseHighMemoryUsage CE S3
  ▼ **Deckhouse memory usage is too high.**

  Deckhouse pod `POD_NAME` on node `NODE_NAME` has high memory usage.

  Please, if possible, get the dumps for debugging purposes (it will take around 30 seconds):

  ```
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o
  jsonpath='{.status.hostIP}'):4222/debug/pprof/heap" -o /tmp/heap.pprof
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o
  jsonpath='{.status.hostIP}'):4222/debug/pprof/goroutine" -o /tmp/goroutine.pprof
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o
  jsonpath='{.status.hostIP}'):4222/debug/pprof/profile?seconds=30" -o /tmp/cpu.pprof
  ```

  and send these files (/tmp/heap.pprof, /tmp/cpu.pprof and /tmp/goroutine.pprof) to the support team.

- DeckhouseMigratedModuleNotDownloaded CE S6
  ▼ **Migrated module MODULE_NAME has not been downloaded yet.**

  Module `MODULE_NAME` specified in DeckhouseRelease was moved to an external source and is available in a ModuleSource registry, but it has not been downloaded yet (no deployed ModuleRelease exists). DeckhouseRelease will be locked until the module is downloaded.

  This alert is usually temporary while the module source controller downloads the module from the registry. If the alert persists, this may indicate:

  - The module's update policy or ModuleSource is misconfigured, preventing the module from being downloaded.
  - A network issue is preventing the module from being downloaded.

  To investigate the issue, do the following:

  - Check the ModuleRelease objects for the module:

    ```
    d8 k get mr -l module=MODULE_NAME
    ```

  - Check the configured ModuleSources:

    ```
    d8 k get modulesources
    ```

  - Check the `deckhouse` logs for more details:

    ```
    d8 k -n d8-system logs -f -l app=deckhouse
    ```

- DeckhouseMigratedModuleNotFound CE S6
  ▼ **Migrated module MODULE_NAME was not found in any ModuleSource registry.**

  Module `MODULE_NAME` specified in DeckhouseRelease was moved to an external source and was not found in any available ModuleSource registry. DeckhouseRelease will be locked until the module becomes available in one of the configured ModuleSource registries.

  This may indicate:

  - A ModuleSource registry is misconfigured.
  - The module has not been published to the registry yet.
  - A network issue is preventing access to the registry.

To investigate the issue, do the following:

- Check the configured ModuleSources:

```
d8 k get modulesources
```

- List the available module sources:

```
d8 k -n d8-system exec -it deployment/deckhouse -c deckhouse -- deckhouse-controller
registry get sources
```

- List the modules available in a specific source:

```
d8 k -n d8-system exec -it deployment/deckhouse -c deckhouse -- deckhouse-controller
registry get modules source_name
```

- Check the `deckhouse` logs for more details:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- DeckhouseModuleUpdatingFailedBrokenSequence CE S4
  ▼ **Module update has failed. Updating sequence is broken.**

  The ‘MODULE_NAME’ module update has failed.

  Current version: XXX. Desired version: VERSION_NUMBER.

  Attempt to download interim releases failed. Deckhouse updates modules one minor version at a time, so every intermediate version must exist in the registry.

  Possible reasons:

  - Some of the necessary versions of the module image are not available in the registry.
  - The registry is unreachable.

  To resolve this issue, follow these steps:

  1. Check which module releases exist and where the sequence stopped:

```
d8 k get modulereleases -l module=MODULE_NAME
```

  2. Read the reason recorded on the release that is not deployed:

```
d8 k get modulereleases -l module=MODULE_NAME -o custom-
columns='NAME:.metadata.name,VERSION:.spec.version,PHASE:.status.phase,MESSA
GE:.status.message'
```

  3. Find the module source that serves this repo. The `registry` label above is a repo URL, not a source name:

```
d8 k get modulesource -o custom-
columns='NAME:.metadata.name,REPO:.spec.registry.repo'
```

  4. Ensure that every minor version of the ‘MODULE_NAME’ module image between XXX and VERSION_NUMBER is available in the registry ‘XXX’. If the cluster uses a mirror, the missing versions have to be pulled into it first.

- DeckhouseModuleUpdatingFailedModuleIsNotValid CE S4
  ▼ **Module update has failed. Module is not valid.**

  The ‘MODULE_NAME’ module update has failed.

  Desired version: VERSION_NUMBER.

  Possible reasons:

  - The module image is corrupted.

  To resolve this issue, ensure that the VERSION_NUMBER image in the registry ‘XXX’ is valid.

Check which versions of the ‘MODULE_NAME’ module are available in the registry:

```
d8 cr ls XXX/MODULE_NAME/release
```

For a private registry, pass the credentials from the corresponding ModuleSource:

```
d8 cr ls -u USERNAME -p PASSWORD XXX/MODULE_NAME/release
```

To get the exact error, check Deckhouse logs: `d8 k -n d8-system logs deploy/deckhouse | grep "failed to download module"`

- DeckhouseReleaseDisruptionApprovalRequired CE S4
  ▼ **Deckhouse release disruption approval required.**

  The new Deckhouse release includes a disruptive update that requires manual approval.

  To check the details, run the following command:

  ```
  d8 k describe DeckhouseRelease NAME
  ```

  To approve the disruptive update, run the following command:

  ```
  d8 k annotate DeckhouseRelease NAME release.deckhouse.io/disruption-approved=true
  ```

- DeckhouseReleaseIsBlocked CE S5
  ▼ **Deckhouse release requirements haven't been met.**

  The requirements for the Deckhouse release haven’t been met.

  To check the details, run the following command:

  ```
  d8 k describe DeckhouseRelease NAME
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S3
  ▼ **A new Deckhouse release is awaiting manual approval.**

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S6
  ▼ **A new Deckhouse release is awaiting manual approval.**

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S9
  ▼ **A new Deckhouse release is awaiting manual approval.**

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseNotificationNotSent CE S4
  ▼ **Deckhouse release notification webhook hasn't been sent.**

  The Deckhouse release notification webhook failed to send.

  To check the notification webhook address, run the following command:

  ```
  d8 k get mc deckhouse -o yaml
  ```

- DeckhouseUpdating CE S4

  Deckhouse is being updated to XXX.

- DeckhouseUpdatingFailed CE S4
  ▼ Deckhouse update has failed.

  The Deckhouse update to version VERSION_NUMBER has failed.

  Possible reasons:

  - The next minor or patch version of the Deckhouse image is not available in the registry.
  - The Deckhouse image is corrupted.
  - Deckhouse cannot reach the registry.

  To resolve this issue, follow these steps:

  1. Check the state of the releases. The one stuck in `Pending` is the one that failed:

     ```
     d8 k get deckhousereleases
     ```

  2. Read the reason recorded on that release:

     ```
     d8 k get deckhousereleases -o custom-
     columns='NAME:.metadata.name,PHASE:.status.phase,MESSAGE:.status.message'
     ```

  3. Read the corresponding error in the logs:

     ```
     d8 k -n d8-system logs -l app=deckhouse --tail=200 | grep -i release
     ```

  4. Ensure that version VERSION_NUMBER of the Deckhouse image is present in the registry. If the cluster uses a mirror, the missing version has to be pulled into it first.

- ModuleAtConflict CE S4
  ▼ Conflict detected for module MODULE_NAME.

  Deckhouse has detected conflicting sources for the MODULE_NAME module: it is available from more than one ModuleSource and no source is selected, so the module is not deployed.

  To resolve this issue, follow these steps:

  1. List the sources that provide the module:

     ```
     d8 k get module MODULE_NAME -o jsonpath='{.properties.availableSources}{"\n"}'
     ```

  2. Compare what each source offers, so you pick the one you actually want:

     ```
     d8 k get modulerelease -l module=MODULE_NAME
     ```

  3. Select the source in the module configuration:

     ```
     d8 k patch moduleconfig MODULE_NAME --type=merge -p '{"spec":
     {"source":"SOURCE_NAME"}}'
     ```

  4. Verify that the module has left the `Conflict` phase. The `SOURCE` and `PHASE` columns show the result:

     ```
     d8 k get module MODULE_NAME
     ```

- ModuleConfigObsoleteVersion CE S4
  ▼ ModuleConfig NAME uses obsolete settings version VERSION_NUMBER, the latest one is XXX.

  Deckhouse has detected that ModuleConfig `NAME` has `spec.version: VERSION_NUMBER`, while the latest settings schema version of the `NAME` module is XXX.

  Deckhouse converts `spec.settings` from version VERSION_NUMBER to version XXX on the fly, so the module keeps working, but the stored configuration remains outdated. Support for version VERSION_NUMBER may be dropped in a future release.

To resolve this issue, follow these steps:

1. Review the migration notes for the `NAME` module settings, one line per conversion step:

   ```
   d8 k get modulesettingsdefinition NAME -o jsonpath='{range
   .spec.versions[*].conversions[*]}{.descriptions.en}{"\n"}{end}'
   ```

2. Migrate `spec.settings` accordingly and set `spec.version` to XXX:

   ```
   d8 k edit moduleconfig NAME
   ```

3. Verify that the version has been updated and no warnings are left:

   ```
   d8 k get moduleconfig NAME -o jsonpath='{.spec.version}{"\t"}{.status.message}
   {"\n"}'
   ```

- ModuleHasDeprecatedUpdatePolicy CE S4
  ▼ Module MODULE_NAME is matched by the deprecated update policy UPDATE_POLICY_NAME.

  Deckhouse has detected that the module `MODULE_NAME` is using the deprecated update policy `UPDATE_POLICY_NAME`. The `v1alpha1` policy has a selector that no longer functions.

  To specify the update policy in the module configuration, run the following command:

  ```
  d8 k patch moduleconfig MODULE_NAME --type='json' -p='[{"op": "add", "path":
  "/spec/updatePolicy", "value": "UPDATE_POLICY_NAME"}]'
  ```

  After resolving all alerts related to the update policy `UPDATE_POLICY_NAME`, clear the selector by running the following command:

  ```
  d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io UPDATE_POLICY_NAME --type='json' -
  p='[{"op": "replace", "path":
  "/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}}]'
  ```

- ModuleIsDeprecated CE S9
  ▼ Module MODULE_NAME is deprecated.

  The module `MODULE_NAME` is deprecated and will not receive new updates. The module may be removed in a future release, so disable it in a controlled manner rather than waiting for it to disappear.

  To resolve this issue, follow these steps:

  1. Check the module state before changing anything. The `STAGE` column reads `Deprecated`, and the conditions show whether anything still runs:

     ```
     d8 k get module MODULE_NAME
     ```

  2. Find out whether other modules declare a dependency on it:

     ```
     d8 k get modules -o json | jq -r '.items[] |
     select(.properties.requirements.modules | has("MODULE_NAME")) |
     .metadata.name'
     ```

  3. Once nothing depends on it and its workloads are no longer needed, disable the module:

     ```
     d8 k patch moduleconfig MODULE_NAME --type=merge -p '{"spec":{"enabled":false}}'
     ```

- ModuleIsInMaintenanceMode CE S6
  ▼ Module MODULE_NAME is running in maintenance mode.

  Module `MODULE_NAME` is running in maintenance mode. In this mode, its state is not reconciled, which prevents configuration or hook changes from being applied automatically.

  To switch the module back to normal mode, edit the corresponding ModuleConfig resource with the following command:

```
d8 k patch moduleconfig MODULE_NAME --type=json -p='[{"op": "remove", "path": "/spec/maintenance"}]'
```

- ModuleReleaseIsBlockedByRequirements CE S6
  ▼ A new release for module MODULE_NAME has been blocked due to unmet requirements.

  A new release for module `MODULE_NAME` has been blocked because it doesn’t meet the required conditions.

  To check the requirements, run the following command:

  ```
  d8 k  get mr NAME -o json | jq .spec.requirements
  ```

- ModuleReleaseIsOutdated CE S5
  ▼ Module MODULE_NAME is XXX or more minor versions behind the latest available release.

  Module `MODULE_NAME` is running XXX or more minor versions behind the latest available release. This module is significantly outdated and may have compatibility issues.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsOutdated CE S7
  ▼ Module MODULE_NAME is 2 minor versions behind the latest available release.

  Module `MODULE_NAME` is running 2 minor versions behind the latest available release. Updating is recommended to maintain compatibility and access to latest features.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsOutdated CE S9
  ▼ Module MODULE_NAME is 1 minor version behind the latest available release.

  Module `MODULE_NAME` is running 1 minor version behind the latest available release. Consider updating to maintain compatibility and security.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsWaitingManualApproval CE S6
  ▼ A new release for module MODULE_NAME is awaiting manual approval.

  A new release for module `MODULE_NAME` is available but requires manual approval before it can be applied.

  To approve the module release, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

Module descheduler

Module documentation

- D8DocumentationModulesListNotBuilt CE S8
  ▼ The page of the list of modules in the documentation is not updated.

  docs-builder could not build the page of the list of modules for 30 minutes. The page shows the list as it was built before, or says that the list is being prepared.

  Check the logs of docs-builder:

```
d8 k -n d8-system logs -l app=documentation -c builder --tail=-1 | grep "write the list
of modules"
```

- D8DocumentationModuleVersionsNotUpdated CE S8
  ▼ The versions of modules in the documentation search were not updated for an hour.

  docs-builder updates the versions of modules the search and the AI export use every 15 minutes; the last update finished more than an hour ago, or none has finished in the hour since docs-builder started reading the registry.

  Check the logs of docs-builder:

  ```
  d8 k -n d8-system logs -l app=documentation -c builder --tail=-1 | grep -i -E
  "resolve|render pinned"
  ```

- D8DocumentationNoLeader CE S7
  ▼ No documentation replica reads the registry.

  No replica of the documentation holds the `docs-builder-ondemand-leader` Lease in `d8-system`. The versions of modules the search uses are not updated; versions users open are still prepared.

  Check the Lease and the logs of docs-builder:

  ```
  d8 k -n d8-system get lease docs-builder-ondemand-leader -o yaml
  d8 k -n d8-system logs -l app=documentation -c builder --tail=-1 | grep -i lead
  ```

- D8DocumentationPinnedVersionsTooLarge CE S8
  ▼ The module versions the documentation keeps at all times take more space than planned.

  docs-builder keeps the versions of modules the search uses on the cache volume of the pod and never evicts them. They take XXXB, more than the space set aside for them.

  Versions users open are evicted sooner and prepared again more often. If the volume fills up, Kubernetes evicts the documentation pod, and the versions are prepared again unless they are kept in S3.

  The size is in the `docs builder ondemand cache pinned_bytes` metric, the space set aside in `docs_builder_ondemand_cache_pinned_budget_bytes`: the cache volume of the pod less 2 GiB of versions users open. The space is not configurable: report the alert to the Deckhouse technical support with the number of modules enabled in the cluster.

- D8DocumentationReplicaReadsRegistry CE S8
  ▼ A documentation replica can not read S3 and reads the registry itself.

  A replica of the documentation could not read the versions of the leader from S3 for 10 minutes and prepares them from the registry itself until S3 answers again.

  Check the S3 storage of the documentation (`moduleStorage.s3` of the `documentation` ModuleConfig) and the logs of docs-builder:

  ```
  d8 k -n d8-system logs -l app=documentation -c builder --tail=-1 | grep -i s3
  ```

- D8DocumentationS3NotReady CE S9
  ▼ Module versions documentation is not kept in S3, the S3 storage is not ready.

  The `documentation` ModuleConfig keeps the prepared documentation of module versions in S3 (`moduleStorage.s3.type`), but the S3 storage has not been usable for 15 minutes. Until it is, the documentation is kept in the documentation pod only and is prepared again after a restart of the pod.

  With `type: External`, check that the Secret from `moduleStorage.s3.external.secretRef` exists and holds the keys listed on the usage page of the module. The reason is in the log of the `builder` container:

  ```
  d8 k -n d8-system logs deploy/documentation -c builder | grep -i s3
  ```

- D8DocumentationSystemS3Disabled CE S9
  ▼ Module versions documentation is not kept in S3, the system-s3 module is disabled.

  `moduleStorage.s3.type: System` is set in the `documentation` ModuleConfig, but the `system-`

`s3` module is disabled. Until it is enabled, the prepared documentation of module versions is kept in the documentation pod only and is prepared again after a restart of the pod.

Enable the `system-s3` module, or set `moduleStorage.s3.type` to `Disabled` in the `documentation` ModuleConfig.

- D8DocumentationVersionNotRenderedAnew CE S9
  ▼ Some module versions in S3 can not be prepared with the current look of the documentation.

  After an update of the documentation look, VALUE module versions kept in S3 could not be prepared again. They open as they were prepared before the update.

  Find them in the logs of docs-builder:

  ```
  d8 k -n d8-system logs -l app=documentation -c builder --tail=-1 | grep "the template can not render the version"
  ```

- ModuleConfigDeprecated CE S9
  ▼ Deprecated deckhouse-web ModuleConfig detected.

  The `deckhouse-web` module has been renamed to `documentation`, and a new `documentation` ModuleConfig is generated automatically.

  Steps to troubleshoot:

  1. Remove the deprecated `deckhouse-web` ModuleConfig from the CI deployment process.
  2. Delete it using the following command:

     ```
     d8 k delete mc deckhouse-web
     ```

Module istio

- D8IstioActiveWaypointsWithAmbientDisabled EE S6
  ▼ Active Istio waypoint exists while ambient mesh is disabled.

  Deckhouse has detected an active waypoint Deployment `DEPLOYMENT_NAME` in namespace `NAMESPACE`, created for WaypointInstance `XXX`, while ambient mesh is disabled.

  This usually happens if ambient mesh was enabled, a WaypointInstance was created, and then ambient mesh was disabled. In this state the waypoint-controller is not running, so it cannot reconcile or clean up WaypointInstance resources.

  To inspect affected resources, run:

  ```
  d8 k -n NAMESPACE get waypointinstance XXX -o yaml
  d8 k -n NAMESPACE get deploy,svc,sa,gateway,pdb,hpa,vpa -l istio.deckhouse.io/waypoint-instance=XXX
  ```

  Recommended remediation:

  1. Temporarily enable ambient mesh.
  2. Wait until waypoint-controller starts.
  3. Delete the affected WaypointInstance or change configuration as needed.
  4. Wait until managed waypoint resources are reconciled as needed or removed.
  5. Disable ambient mesh again.

  If ambient mesh cannot be re-enabled, manually remove the managed waypoint resources and then remove the WaypointInstance finalizer only after verifying that no managed resources remain.

- D8IstioActualDataPlaneVersionNotEqualDesired EE S8
  ▼ There are pods with Istio data plane version VERSION_NUMBER, but desired version is XXX.

  There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, while the desired version is `XXX`. As a result, the Istio version will be changed after the pod is restarted.

  To resolve the issue, use the following cheat sheet:

```
### Namespace-wide configuration
# `istio.io/rev=vXYZ`: Use a specific revision.
# `istio-injection=enabled`: Use the global revision.
d8 k get ns NAMESPACE --show-labels

### Pod-wide configuration
d8 k -n NAMESPACE get pods -l istio.io/rev=DESIRED_VISION
```

- D8IstioActualVersionIsNotInstalled EE S4
  ▼ **The control plane version for pods with injected sidecars isn't installed.**

  There are pods in the `NAMESPACE` namespace with injected sidecars of version `VERSION_NUMBER` (revision `REVISION_NUMBER`), but the corresponding control plane version is not installed. As a result, these pods have lost synchronization with the state in Kubernetes.

  To resolve this issue, install the required control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

  To identify orphaned pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -l 'service.istio.io/canonical-name' -o json | jq --arg revision REVISION_NUMBER '.items[] |
  select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
  .revision == $revision) | .metadata.name'
  ```

- D8IstioAdditionalControlplaneDoesntWork CE S4
  ▼ **Additional control plane isn't working.**

  Deckhouse has detected that the additional Istio control plane `ISTIO_REVISION_LABEL` isn’t working.

  As a result, sidecar injection for pods with `ISTIO_REVISION_LABEL` isn’t working as well.

  To check the status of the control plane pods, run the following command:

  ```
  d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
  ```

- D8IstioAllianceAmbientGatewayAddressUnusable EE S6
  ▼ **The ambient east-west gateway has no address peers can dial.**

  The ambient east-west gateway is deployed in this cluster, but publishes no address peer clusters can dial: its Service is missing, it has no external address, no node is running a gateway pod, or every address it has is neither an IP address nor a usable DNS name. The `metadata-exporter` pod logs in `d8-istio` say which.

  Peer clusters keep using the addresses they fetched last, if any. Those that have none cannot reach ambient-mode workloads here.

  To fix, set `alliance.ambientGateway.advertise` to the IP address or DNS name peers should dial. Or make the load balancer assign an address (`alliance.ambientGateway.loadBalancerClass`, `.serviceAnnotations`). Or switch `alliance.ambientGateway.inlet` to `NodePort` and check that gateway pods are scheduled (`.nodeSelector`, `.tolerations`). Or, if ambient multicluster is not needed, switch all multicluster members back to sidecar mode.

- D8IstioAllianceDataPlaneNotConnected EE S4
  ▼ **Data-plane connectivity check failed for XXX NAME.**

  The data-plane health check of XXX `NAME` with uuid `XXX` has been failing for more than 5 minutes.

  This means that actual service mesh traffic between clusters is not flowing properly, even though the control plane metadata exchange may be working.

  Possible causes:

  - The remote cluster’s `alliance-healthcheck` pod is not running.
  - Istio sidecar injection is not working properly on the healthcheck pods.
  - The ingress gateway between clusters is misconfigured or unreachable.
  - mTLS or trust domain configuration issues between clusters.

  To check the status:

```
d8 k get XXX NAME -o json | jq '.status.conditions[] |
select(.type=="DataplaneConnectionReady")'
```

- D8IstioAllianceDiscoveryMetadataEndpointRequestedViaDeprecatedSubdomain EE S5
  ▾ **A peer cluster requests private metadata using a deprecated subdomain.**

  Remote XXX `NAME`, (UUID: `XXX`) requests private metadata from hostname `XXX`, that hostname uses the deprecated `istio` subdomain.

  Update the remote cluster XXX resource: set `spec.metadataEndpoint` to `(publicDomainTemplate + istio-metadata)/metadata/`.

  Migration steps on the remote cluster:

  1. Ensure DNS resolves the new target `(publicDomainTemplate + istio-metadata)`.
  2. Update `spec.metadataEndpoint` to use the new domain name. Keep the path `/metadata/` unchanged.
  3. Verify remote metadata fetch and alliance health after the change.
- D8IstioAllianceIngressGatewayAddressUnusable EE S6
  ▾ **The ingress gateway for federation and multicluster has no address peers can dial.**

  The ingress gateway that peer clusters reach this cluster through is deployed, but publishes no address they can dial: its Service is missing, it has no external address, or no node is running a gateway pod. The `metadata-exporter` pod logs in `d8-istio` say which.

  Peer clusters keep using the addresses they fetched last, if any. Those that have none cannot reach federated services or, over multicluster, sidecar-mode workloads here.

  To fix, set `alliance.ingressGateway.advertise` to the addresses peers should dial. Or make the load balancer assign an address (`alliance.ingressGateway.loadBalancerClass`, `.serviceAnnotations`). Or switch `alliance.ingressGateway.inlet` to `NodePort` and check that gateway pods are scheduled (`.nodeSelector`, `.tolerations`).

- D8IstioConfigAnalysisConflictingGateways EE S5
  ▾ **Conflicting Istio Gateways detected (IST0145).**

  The Istio configuration analyzer found conflicting Gateways in revision `REVISION_NUMBER`.

  Code: `XXX` (`ConflictingGateways`) Namespace: `NAMESPACE` Resource: `XXX`

  Two or more Gateways share the same selector, port, and host match. Ingress traffic may be routed unpredictably or return 404.

  Steps to resolve:

  1. List Gateways and compare `spec.selector`, server ports, and hosts for overlaps involving `XXX`.
  2. Deduplicate or narrow hosts/ports so each combination is unique.
  3. See https://istio.io/latest/docs/reference/config/analysis/ist0145/

  List matching issues in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{type="ConflictingGateways",revision="REVISION_NUMBER"}
  ```

  Check analyzer logs:

  ```
  d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
  ```

- D8IstioConfigAnalysisError EE S5
  ▾ **Istio configuration analysis detected an error-level issue.**

  The Istio configuration analyzer found an error in revision `REVISION_NUMBER`.

  Type: `ERROR_TYPE` (`XXX`) Namespace: `NAMESPACE` Resource: `XXX`

  Inspect the reported resource in namespace `NAMESPACE`.

  Documentation for analysis codes: https://istio.io/latest/docs/reference/config/analysis/

  List analysis issues in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{revision="REVISION_NUMBER",severity="Error"}
  ```

Check analyzer pods and logs:

```
d8 k -n d8-istio get pods -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER
d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
```

- D8IstioConfigAnalysisImageAutoWithoutInjection EE S5
  ▾ **Pod uses image auto without Istio injection (IST0147).**

  The Istio configuration analyzer found pods with `image: auto` that are not targeted for sidecar injection in revision `REVISION_NUMBER`.

  Code: `XXX` (`ImageAutoWithoutInjectionError`) Namespace: `NAMESPACE` Resource: `XXX`

  `image: auto` is rewritten only when the Istio injector mutates the pod. Without injection the container image stays literally `auto` and the pod will not run correctly.

  Steps to resolve:

  1. Enable sidecar injection for the namespace or workload (`istio-injection=enabled` / `istio.io/rev=REVISION_NUMBER`, or a matching RevisionTag).
  2. Restart / recreate the affected pods after injection is enabled.
  3. See https://istio.io/latest/docs/reference/config/analysis/ist0147/

  List matching issues in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{type="ImageAutoWithoutInjectionError",revision="REVISION_NUMBER"}
  ```

  Check analyzer logs:

  ```
  d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
  ```

- D8IstioConfigAnalysisInvalidGatewayCredential EE S5
  ▾ **Gateway TLS credential is invalid (IST0161).**

  The Istio configuration analyzer found an invalid Gateway credential in revision `REVISION_NUMBER`.

  Code: `XXX` (`InvalidGatewayCredential`) Namespace: `NAMESPACE` Resource: `XXX`

  The Secret referenced by the Gateway for TLS is missing or does not contain a usable certificate/key. HTTPS listeners on that Gateway will fail.

  Steps to resolve:

  1. Open the Gateway `XXX` in namespace `NAMESPACE` and note `credentialName` / TLS secret references.
  2. Verify the Secret exists in the same namespace as the ingress Gateway workload and has valid `tls.crt` / `tls.key` (or the expected keys).
  3. See https://istio.io/latest/docs/reference/config/analysis/ist0161/

  List matching issues in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{type="InvalidGatewayCredential",revision="REVISION_NUMBER"}
  ```

  Check analyzer logs:

  ```
  d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
  ```

- D8IstioConfigAnalysisReferencedResourceNotFound EE S5
  ▾ **Istio resource references a missing dependency (IST0101).**

  The Istio configuration analyzer found a missing reference in revision `REVISION_NUMBER`.

  Code: `XXX` (`ReferencedResourceNotFound`) Namespace: `NAMESPACE` Resource: `XXX`

  An Istio object points to a Gateway, Service, ServiceEntry, or another dependency that does not exist (typo, wrong namespace, or resource not created yet). Traffic to that route may fail.

  Steps to resolve:

  1. Inspect the reported resource in namespace `NAMESPACE` and fix the broken reference (use FQDN for cross-namespace hosts, e.g. `reviews.backend.svc.cluster.local`).
  2. Confirm the target object exists.

3. See https://istio.io/latest/docs/reference/config/analysis/ist0101/

List matching issues in Prometheus (Grafana → Explore):

```
d8_istio_config_analysis_issue{type="ReferencedResourceNotFound",revision="REVISION_NUMBER"}
```

Check analyzer logs:

```
d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
```

- D8IstioConfigAnalysisSchemaValidationError EE S5
  ▼ Istio resource failed schema validation (IST0106).

  The Istio configuration analyzer found a schema validation error in revision `REVISION_NUMBER`.

  Code: `XXX` (`SchemaValidationError`) Namespace: `NAMESPACE` Resource: `XXX`

  The resource does not match the Istio API schema and may be ignored or rejected by the control plane.

  Steps to resolve:

  1. Inspect `XXX` in namespace `NAMESPACE` and compare fields with the Istio API for your control plane version.
  2. Fix invalid or unknown fields, then re-apply the manifest.
  3. See https://istio.io/latest/docs/reference/config/analysis/ist0106/

  List matching issues in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{type="SchemaValidationError",revision="REVISION_NUMBER"}
  ```

  Check analyzer logs:

  ```
  d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
  ```

- D8IstioConfigAnalysisWarning EE S4
  ▼ Istio configuration analysis detected a warning-level issue.

  The Istio configuration analyzer found a warning in revision `REVISION_NUMBER`.

  Type: `ERROR_TYPE` (`XXX`) Namespace: `NAMESPACE` Resource: `XXX`

  Inspect the reported resource in namespace `NAMESPACE`.

  Documentation for analysis codes: https://istio.io/latest/docs/reference/config/analysis/

  List analysis warnings in Prometheus (Grafana → Explore):

  ```
  d8_istio_config_analysis_issue{revision="REVISION_NUMBER",severity="Warning"}
  ```

  Check analyzer pods and logs:

  ```
  d8 k -n d8-istio get pods -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER
  d8 k -n d8-istio logs -l app=istio-config-analyzer,istio.io/rev=REVISION_NUMBER --tail=100
  ```

- D8IstioCustomCAMaterialInvalid CE S4
  ▼ The Istio custom CA material is invalid but is being served to istiod.

  The Istio module is publishing CA material (`XXX`) that failed validation to istiod.

  The module never hard-blocks on module-owned or last-good material (doing so would rotate the mesh root and break mTLS cluster-wide), so it keeps serving this material with only a warning. However, istiod may reject it, which can prevent it from issuing or renewing workload certificates.

  Depending on the `source`:

  - `cacerts` — the `d8-istio/cacerts` Secret (or the CA reused from it) is not a valid CA. Inspect the Secret and restore valid `ca-cert.pem`/`ca-key.pem` (and `root-cert.pem`/`cert-chain.pem` if used).
  - `secretRef:namespace/name` — the last-good CA that was previously resolved from this `ca.secretRef` has become invalid (for example, its chain expired). Fix the referenced Secret so the next re-resolution (every 5 minutes) succeeds.

Refer to the custom CA documentation.

- D8IstioCustomCASecretRefUnresolved CE S5
  ▼ **The Istio custom CA ca.secretRef cannot be resolved.**

  The Istio module could not re-resolve the configured `ca.secretRef` (`XXX`) and is keeping the last successfully-resolved CA.

  The mesh is not regressed — istiod keeps running on the exact CA material it already had — but any change made to the source Secret is not being picked up. Common causes:

  - the referenced Secret was deleted or renamed;
  - the referenced Secret was made malformed (not valid PEM, not a CA certificate, or a mismatched key pair);
  - a transient Kubernetes API problem.

  The module re-resolves the Secret every 5 minutes; this alert clears automatically once resolution succeeds. Inspect the referenced Secret and restore valid CA material.

  Refer to the custom CA documentation.

- D8IstioDataPlaneVersionMismatch EE S8
  ▼ **There are pods with data plane version different from the control plane version.**

  There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, which is different from the control plane version `DESIRED_VERSION`.

  Steps to resolve the issue:

  1. Restart affected pods and use the following PromQL query to get a full list:

     ```
     max by (namespace, dataplane_pod)
     (d8_istio_dataplane_metadata{full_version="VERSION_NUMBER"})
     ```

  2. Use the automatic Istio data plane upgrade.

- D8IstioDataPlaneWithoutIstioInjectionConfigured EE S4
  ▼ **Detected pods with Istio sidecars but istio-injection isn't configured.**

  There are pods in the `NAMESPACE` namespace with Istio sidecars, but istio-injection isn’t configured. As a result, these pods will lose their Istio sidecars after being recreated.

  To identify the affected pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -o json | jq -r --arg revision REVISION_NUMBER '.items[] |
  select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
  .revision == $revision) | .metadata.name'
  ```

- D8IstioDeprecatedIstioVersionInstalled CE
  ▼ **The installed Istio version has been deprecated.**

  Deckhouse has detected that a deprecated Istio version `VERSION_NUMBER` is installed.

  Support for this version will be removed in upcoming Deckhouse releases. The higher the alert severity, the greater the probability of support being discontinued.

  To learn how to upgrade Istio, refer to the upgrade guide.

- D8IstioDesiredVersionIsNotInstalled EE S6
  ▼ **Desired control plane version isn't installed.**

  There is a desired Istio control plane version `XXX` (revision `REVISION_NUMBER`) configured for pods in the `NAMESPACE` namespace, but that version isn’t installed. As a result, pods can’t be recreated in the `NAMESPACE` namespace.

  To resolve this issue, install the desired control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

  Use the following cheat sheet:

```
### Namespace-wide configuration
# `istio.io/rev=vXYZ`: Use a specific revision.
# `istio-injection=enabled`: Use the global revision.
d8 k get ns NAMESPACE --show-labels

### Pod-wide configuration
d8 k -n NAMESPACE get pods -l istio.io/rev=REVISION_NUMBER
```

- D8IstioFederationMetadataEndpointDoesntWork EE S6
  ▼ IstioFederation FEDERATION_NAME gets no usable metadata from the remote cluster.

  The Deckhouse hook could not get usable metadata from the endpoint `ENDPOINT_NAME` for IstioFederation `FEDERATION_NAME`.

  The reason and the details are in the `PublicMetadataExchangeReady` and `PrivateMetadataExchangeReady` conditions of the IstioFederation. The endpoint may respond normally while the metadata is still unusable, for example, when the ingress gateway of the remote cluster has no address yet.

  To view the conditions, run the following command:

  ```
  d8 k get istiofederation FEDERATION_NAME -o jsonpath='{range .status.conditions[*]}
  {.type}: {.status}: {.reason}: {.message}{"\n"}{end}'
  ```

  To reproduce the request to the public endpoint, run the following command:

  ```
  curl ENDPOINT_NAME
  ```

  To reproduce the request to private endpoints (run from the Deckhouse pod), run the following:

  ```
  KEY="$(deckhouse-controller module values istio -o json | jq -r
  .internal.remoteAuthnKeypair.priv)"
  LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
  .global.discovery.clusterUUID)"
  REMOTE_CLUSTER_UUID="$(d8 k get istiofederation FEDERATION_NAME -o json | jq -r
  .status.metadataCache.public.clusterUUID)"
  TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
  iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
  claim scope=private-federation --ttl 1h)"
  curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
  ```

- D8IstioFederationRemoteClusterUUIDMismatch EE S4
  ▼ IstioFederation peer cluster UUID changed.

  Public metadata from `ENDPOINT_NAME` for IstioFederation `FEDERATION_NAME` reports cluster UUID `XXX`, but the previously pinned UUID is `XXX`.

  Deckhouse stops private metadata exchange and does not send a federation JWT until the peer identity is confirmed again.

  Check whether `spec.metadataEndpoint` was pointed to another cluster or the remote endpoint was replaced.

  To inspect the resource:

  ```
  d8 k get istiofederation FEDERATION_NAME -o yaml
  curl ENDPOINT_NAME
  ```

  If the peer change is intentional, recreate the IstioFederation resource so a new UUID can be pinned.

- D8IstioGlobalControlplaneDoesntWork CE S4
  ▼ Global control plane isn't working.

  Deckhouse has detected that the global Istio control plane `ISTIO_REVISION_LABEL` isn’t working.

  As a result, sidecar injection for pods with global revision isn’t working as well, and the validating webhook for Istio resources is absent.

  To check the status of the control plane pods, run the following command:

```
d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
```

- D8IstioMulticlusterMetadataEndpointDoesntWork EE S6
  ▼ IstioMulticluster MULTICLUSTER_NAME gets no usable metadata from the remote cluster.

  The Deckhouse hook could not get usable metadata from the endpoint `ENDPOINT_NAME` for IstioMulticluster `MULTICLUSTER_NAME`.

  The reason and the details are in the `PublicMetadataExchangeReady` and `PrivateMetadataExchangeReady` conditions of the IstioMulticluster. The endpoint may respond normally while the metadata is still unusable, for example, when the remote cluster does not deploy the gateway this cluster reaches it through.

  To view the conditions, run the following command:

  ```
  d8 k get istiomulticluster MULTICLUSTER_NAME -o jsonpath='{range .status.conditions[*]}
  {.type}: {.status}: {.reason}: {.message}{"\n"}{end}'
  ```

  To reproduce the request to the public endpoint, run the following command:

  ```
  curl ENDPOINT_NAME
  ```

  To reproduce the request to private endpoints (run from the `d8-system/deckhouse` pod), run the following:

  ```
  KEY="$(deckhouse-controller module values istio -o json | jq -r
  .internal.remoteAuthnKeypair.priv)"
  LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
  .global.discovery.clusterUUID)"
  REMOTE_CLUSTER_UUID="$(d8 k get istiomulticluster MULTICLUSTER_NAME -o json | jq -r
  .status.metadataCache.public.clusterUUID)"
  TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
  iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
  claim scope=private-multicluster --ttl 1h)"
  curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
  ```

- D8IstioMulticlusterRemoteAPIHostDoesntWork EE S6
  ▼ Multicluster remote API host health check failed.

  The remote API host `API_HOST` for IstioMulticluster `MULTICLUSTER_NAME` has failed the health check performed by the Deckhouse monitoring hook.

  To reproduce the request (run from the `d8-system/deckhouse` pod), run the following:

  ```
  TOKEN="$(deckhouse-controller module values istio -o json | jq -r --arg ah API_HOST
  '.internal.multiclusters[]| select(.apiHost == $ah)| .apiJWT ')"
  curl -H "Authorization: Bearer $TOKEN" https://API_HOST/version
  ```

- D8IstioMulticlusterRemoteClusterUUIDMismatch EE S4
  ▼ IstioMulticluster peer cluster UUID changed.

  Public metadata from `ENDPOINT_NAME` for IstioMulticluster `MULTICLUSTER_NAME` reports cluster UUID `XXX`, but the previously pinned UUID is `XXX`.

  Deckhouse stops private metadata exchange and does not send a multicluster JWT until the peer identity is confirmed again.

  Check whether `spec.metadataEndpoint` was pointed to another cluster or the remote endpoint was replaced.

  To inspect the resource:

  ```
  d8 k get istiomulticluster MULTICLUSTER_NAME -o yaml
  curl ENDPOINT_NAME
  ```

  If the peer change is intentional, recreate the IstioMulticluster resource so a new UUID can be pinned.

- D8IstioNetworkNameIsNotLabelValue CE S6
  ▼ **The Istio network name is not a valid Kubernetes label value.**

  The Istio network name `XXX` of this cluster is not a valid Kubernetes label value. Because of this, the name is left out of every place where a label value is required.

  Contact Deckhouse technical support to resolve the issue.

- D8IstioOperatorReconcileError CE S5
  ▼ **The istio-operator is unable to reconcile Istio control plane setup.**

  Deckhouse has detected an error in the `istio-operator` reconciliation loop.

  To investigate the issue, check the operator logs:

  ```
  d8 k -n d8-istio logs -l app=operator,revision=REVISION_NUMBER
  ```

- D8IstioPodsWithoutIstioSidecar EE S4
  ▼ **Detected pods without Istio sidecars but with istio-injection configured.**

  There is a pod `POD_NAME` in the `NAMESPACE` namespace without Istio sidecars, but with istio-injection configured.

  To identify the affected pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -l '!service.istio.io/canonical-name' -o json | jq -r
  '.items[] | select(.metadata.annotations."sidecar.istio.io/inject" != "false")
  | .metadata.name'
  ```

- D8IstioRemoteClusterNotSynced EE S4
  ▼ **XXXIstioMulticluster MULTICLUSTER_NAME is not synchronized.XXXIstio remote cluster CLUSTER_ID is not synchronized.XXX**

  XXXThe Istio control plane instance `INSTANCE NAME` cannot synchronize with the remote cluster described by IstioMulticluster `MULTICLUSTER_NAME` (cluster ID `CLUSTER_ID`).XXXThe Istio control plane instance `INSTANCE_NAME` cannot synchronize with the remote cluster `CLUSTER_ID`.XXX

  Possible causes:

  - The remote cluster is offline.
  - The remote API endpoint is not reachable.
  - The remote ServiceAccount token is invalid or expired.
  - There is a TLS or certificate issue between the clusters.
  - Remote and local Istio versions mismatch.

  XXXCheck the conditions of the IstioMulticluster:

  ```
  d8 k get istiomulticluster MULTICLUSTER_NAME -o jsonpath='{range .status.conditions[*]}
  {.type}: {.status}: {.reason}: {.message}{"\n"}{end}'
  ```

  All conditions must be `True`. If a condition is not `True`, its `reason` and `message` say what is wrong. Access to the remote API is reported by the `RemoteAPIServerReady` condition.

  If there is no such resource, this cluster was registered by the secret `XXX` alone — check the API address and the token in the kubeconfig it stores. XXXNo IstioMulticluster resource registered this cluster, so check the API address and the token in the kubeconfig stored in the secret `XXX`. XXX

- D8IstioReservedUIDConflict CE S4
  ▼ **Application container is running as UID 1337 (reserved by Istio).**

  Container `CONTAINER_NAME` in pod `POD_NAME` in namespace `NAMESPACE` is running as UID `1337`.

  UID `1337` is reserved by Istio for the `istio-proxy` sidecar container. When application containers run with this UID, their traffic bypasses Istio completely — no routing rules, mTLS, or telemetry will be applied.

  To fix the issue, change the `runAsUser` in the container's or pod's `securityContext` to a different UID.

- D8IstioVersionIsIncompatibleWithK8sVersion CE S3
  ▼ **The installed Istio version is incompatible with the Kubernetes version.**

  The installed Istio version `VERSION_NUMBER` may not work properly with the current Kubernetes

version `VERSION_NUMBER` because it’s not supported officially.

To resolve the issue, upgrade Istio following the guide.

- IstioIrrelevantExternalServiceFound CE S5
  ▼ **External service found with irrelevant ports specifications.**

  A service `NAME` in the `NAMESPACE` namespace has an irrelevant port specification.

  The `.spec.ports[]` field isn’t applicable for services of the `ExternalName` type. However, Istio renders port listeners for external services as `0.0.0.0:port`, which captures all traffic to the specified port. This can cause problems for services that aren’t registered in the Istio registry.

  To resolve the issue, remove the `.spec.ports` section from the service configuration. It is safe.

Module kube-dns

- KubernetesCoreDNSHasCriticalErrors CE S5
  ▼ **Critical errors found in CoreDNS.**

  Deckhouse has detected at least one critical error in the CoreDNS pod POD_NAME.

  To resolve the issue, review the container logs:

  ```
  d8 k -n kube-system logs POD_NAME
  ```

Module metallb

- D8MetallbBGPSessionDown EE S4
  ▼ **MetalLB BGP session is down.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME has BGP session PEER down.

  Check the logs for details:

  ```
  d8 k -n d8-metallb logs daemonset/speaker -c speaker
  ```

- D8MetallbConfigNotLoaded EE S4
  ▼ **The MetalLB configuration hasn't been loaded.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME hasn’t been loaded.

  To find the cause of the issue, review the controller logs:

  ```
  d8 k -n d8-metallb logs deploy/controller -c controller
  ```

- D8MetallbConfigStale EE S4
  ▼ **MetalLB is running on a stale configuration.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME is running on a stale configuration because the latest configuration failed to load.

  To find the cause of the issue, review the controller logs:

  ```
  d8 k -n d8-metallb logs deploy/controller -c controller
  ```

- D8MetallbNotSupportedServiceAnnotationsDetected SE S4
  ▼ **The annotation 'ANNOTATION_NAME' has been deprecated for the service 'NAME' in the 'NAMESPACE' namespace.**

  The annotation ‘ANNOTATION_NAME’ has been deprecated for the service ‘NAME’ in the ‘NAMESPACE’ namespace.

  The following service annotations are no longer effective:

  - `metallb.universe.tf/ip-allocated-from-pool`: Remove this annotation.
  - `metallb.universe.tf/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate MetalLoadBalancerClass.
  - `metallb.universe.tf/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips: IP`.
  - `metallb.universe.tf/allow-shared-ip`: Replace it with `network.deckhouse.io/load-`

`balancer-shared-ip-key`.
  - `metallb.io/ip-allocated-from-pool`: Remove this annotation.
  - `metallb.io/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate MetalLoadBalancerClass.
  - `metallb.io/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips: IP`.
  - `metallb.io/allow-shared-ip`: Replace it with `network.deckhouse.io/load-balancer-shared-ip-key`.

Please note. Existing LoadBalancer services of Deckhouse have been migrated automatically, but the new ones will not be.

- D8MetallbObsoleteLayer2PoolsAreUsed SE S7
  ▼ **The metallb module has obsolete layer2 pools configured.**

  In ModuleConfig version 2, addressPool ‘NAME’ of type “layer2” are ignored. They should be removed from the configuration.

- D8MetallbUpdateMCVersionRequired SE S5
  ▼ **The metallb ModuleConfig settings are outdated.**

  D8 MetalLB settings are outdated.

  To resolve this issue, increase version in the ModuleConfig `metallb`.

- L2LoadBalancerOrphanServiceFound SE S4
  ▼ **Orphaned service with an irrelevant L2LoadBalancer name has been found.**

  The cluster contains an orphaned service `NAME` in the `NAMESPACE` namespace with an irrelevant L2LoadBalancer name.

  To resolve this issue, verify the L2LoadBalancer name specified in the annotation `network.deckhouse.io/l2-load-balancer-name`.

Module multitenancy-manager

- ClusterResourceGrantPolicyViolation CE S6
  ▼ **Objects in namespace XXX use cluster-wide resources that are not available to its project.**

  Objects in namespace `XXX` reference cluster-wide resources (for example, a StorageClass or a ClusterRole) that the current ClusterResourceGrantPolicy objects do not make available to the project of this namespace. The `project` label of the metric holds the namespace, not the project name. Such an object was created before the policy changed. It keeps working, and when it is updated, only new values are checked.

  Steps to troubleshoot:

  1. Find the objects, their fields and the resources on the “Security” → “Cluster Resource Grant Violations” Grafana dashboard, or in the `d8 cluster objects grant violated{project="XXX"}` metric (labels `violating_object_name`, `violating_field`, `violating_resource` and `grant`).
  2. Check which resources are available to the project:

     ```
     d8 k get available RESOURCE_NAME -n XXX -o yaml
     ```

  3. Make the resource available to the project in a ClusterResourceGrantPolicy, or change the field of the object to an available value.
- D8NamespaceNotAdopted CE S6
  ▼ **Namespace XXX belongs to no project.**

  Namespace `XXX` was created directly more than 10 minutes ago and is still not a project. The multitenancy-manager controller turns every namespace outside the system ones into a project of its own within seconds, whatever the `allowNamespacesWithoutProjects` parameter says. Until it does, the objects created in the namespace skip the checks of a project: the availability of cluster-wide resources, the project template with its network and pod security policies, and the quota.

  The controller has no leader, fails to adopt the namespace, or has adopted it into a Project that does not reconcile. Check its pods and logs:

```
d8 k -n d8-multitenancy-manager get pods
d8 k -n d8-multitenancy-manager logs deploy/multitenancy-manager -c multitenancy-manager --
tail=200 | grep XXX
```

If the controller fails to adopt the namespace, it tries again with a delay of up to 30 seconds, so the namespace becomes a project and the alert resolves soon after the cause is gone, for example an admission policy or webhook that refuses the Project. Once the controller adopts the namespace, check the objects created in it in the meantime.

If a Project named like the namespace exists already, the controller takes the adoption as done and does not try again, and the logs of the namespace controller show nothing. The project may be in `Error`, for example with the `HelmOwnership` condition when the namespace belongs to another Helm release, and then it never labels the namespace: until the conflict is removed, this Project is not updated at all. Check the project:

```
d8 k get project XXX -o jsonpath='{.status.state}{"\n"}{range .status.conditions[*]}
{.type}={.status} {.reason}: {.message}{"\n"}{end}'
```

Remove the cause the condition names, for example the foreign release or its `meta.helm.sh/*` annotations on the namespace, and the project labels the namespace. This also happens to the namespaces that existed before the upgrade.

- D8ProjectQuotaExceeded CE S7
  ▾ **Project XXX has a quota set below its usage.**

  The usage of project `XXX`, summed over all of its namespaces, is above the limits of its quota: the limits were set below the usage, by a change of the quota or when it was created (the `HardLowered` reason of the `Exceeded` condition).

  Nothing running is evicted. Creating more of the resources whose limits are exceeded is denied until the usage is back within the limits.

  1. See which limits are exceeded and by how much:

     ```
     d8 k get projectquotastatus XXX -o jsonpath='{range .status.conditions[*]}{.type}=
     {.status} {.reason}: {.message}{"\n"}{end}'
     ```

  2. See which namespaces use the most (the usage of every namespace is in its `d8-project-quota` ResourceQuota):

     ```
     d8 k get projectquotastatus XXX -o jsonpath='{range .status.topNamespaces[*]}
     {.name}: {.used}{"\n"}{end}'
     d8 k get resourcequota -A -l projects.deckhouse.io/project=XXX
     ```

  3. Raise the limits in `spec.quota` of the Project, or delete what the project does not need.

- D8ProjectQuotaLeaderFlapping CE S5
  ▾ **The leader of the project quota ledger changes too often.**

  The lease of the ledger changed hands more than three times in ten minutes; every change refuses counted creates for a second or more. The `reason` label of `d8_multitenancy_quota_ledger_handover_total` says why: `expired` points at late renewals (the API server, the `leader-election` priority level), `stepdown` at the canary, `graceful` at restarts of the pods (rollouts, evictions).

- D8ProjectQuotaLedgerCanaryFailing CE S4
  ▾ **Most API servers do not reach the leader of the project quota ledger.**

  The canary writes of the ledger do not reach its webhook through most of the API servers. The leader hands the lease over when another replica is ready; this alert says it could not, or that the new leader is not reached either. Check the network from the master nodes to the pods of `d8-multitenancy-manager` and the CNI on the masters. The steps of `D8ProjectQuotaLedgerDown` apply.

- D8ProjectQuotaLedgerDown CE S4
  ▾ **No replica of the project quota ledger answers.**

  No replica of the `quota-ledger` container of the `multitenancy-manager` module holds its lease with synced informers. Creating the objects a project quota counts in the namespaces of projects with a quota fails with `project quota ledger unavailable`, or with `failed calling webhook`

`"quota-ledger.multitenancy-manager.deckhouse.io"`. Deletions, running objects and other namespaces are not affected.

1. Check the pods, the lease and the route:

   ```
   d8 k -n d8-multitenancy-manager get pods,lease,endpointslices
   d8 k -n d8-multitenancy-manager logs deploy/multitenancy-manager -c quota-ledger --tail=200
   ```

   The logs say whether the ledger is `fenced` (its lease renewals are late) or `syncing`.

2. If the cause cannot be removed quickly, delete the webhook as `system:sudouser`. While no replica holds the route, the API servers reach none of them, so suspending the ledger with the `quota-ledger-suspend` ConfigMap does not help here. The sum check stops at once; Deckhouse puts the webhook back within about 4 minutes, so this buys time rather than switching the check off. Disabling the `multitenancy-manager` module removes the check together with the rest of the module.

   ```
   d8 k --as=system:sudouser delete validatingwebhookconfiguration d8-multitenancy-manager-quota-ledger
   ```

- D8ProjectQuotaLedgerDrift CE S6
  ▼ **The project quota ledger differs from the API server.**

  The ledger holds the usage of some quota differently from a quorum read of the same version of it. The state of the ledger is inconsistent. Save the logs of the `quota-ledger` container, then restart the pod of the leader once so that another replica takes over. Do not restart it in a loop.

- D8ProjectQuotaLedgerSlow CE S5
  ▼ **The project quota ledger answers slowly.**

  The 99th percentile of the calls of the ledger webhook, as the API servers see them, is above 100 ms. The ledger answers within 25 ms of its own, so the delay comes from the network or from the CPU of the `quota-ledger` container. Check the CPU of the master nodes and of the container, the `d8_multitenancy_quota_ledger_inflight` and `d8_multitenancy_quota_ledger_requests_total{result="busy"}` series, and other webhooks on `resourcequotas/status`.

- D8ProjectQuotaLedgerUnreachableFromAPIServer CE S5
  ▼ **The API server XXX does not reach the project quota ledger.**

  The canary writes through the API server `XXX` fail on the webhook of the ledger, while the other API servers reach it. Counted creates that land on this API server in the namespaces of projects with a quota wait for the webhook timeout and fail. Check the network of that master node to the pods of `d8-multitenancy-manager`; the webhooks of other modules are likely affected the same way.

- D8ProjectQuotaNotEnforced CE S6
  ▼ **The quota of project XXX does not apply in every namespace of the project.**

  The `Enforced` condition of the ProjectQuotaStatus of project `XXX` has been `False` for 15 minutes for one of these reasons:

  - `KeyRollout`: the `d8-project-quota` ResourceQuota of some namespace of the project is missing or not calculated with every limit of the quota. The resource quota controller of kube-controller-manager calculates a new quota within seconds, so a quota that stays uncalculated points at kube-controller-manager or at the multitenancy-manager controller; the message of the condition names the namespaces and the error of the controller, if any. Until it is calculated, the objects of users are refused in a namespace labelled `projects.deckhouse.io/quota: pending`. In a namespace labelled `projects.deckhouse.io/quota: enforced`, the objects the quota counts are refused only while its `d8-project-quota` has no calculated limit at all or has scopes; a limit added to a calculated quota holds nothing back.
  - `KeyNotServed`: some keys of the quota count nothing, because a `count/` key names a resource of an API group that the API server does not serve, or because the module that serves the resources of a key, such as `virtualization`, is disabled. Enable that module, or remove the key.
  - `WebhookMissing`: the `d8-multitenancy-manager-quota-ledger` webhook is missing or its `caBundle` is not the CA of the `quota-ledger-certs` Secret, so the sum is not checked. Deckhouse puts it back on the next run of the module; see the `D8ProjectQuotaWebhookMisconfigured` alert.
  - `WebhookMisconfigured`: the `d8-multitenancy-manager-quota-ledger` webhook is changed (its rules, `namespaceSelector`, `objectSelector`, `matchConditions`, `failurePolicy` or service) so that writes of `d8-project-quota` do not reach the ledger.

The message of the condition names the field. Deckhouse puts the webhook back on the next run of the module; see the `D8ProjectQuotaWebhookMisconfigured` alert.
- `WebhookNotCalled`: the canary of the ledger found API servers that admit objects without calling the `d8-multitenancy-manager-quota-ledger` webhook, so the sum is not checked for what they admit. The message of the condition names them; check the rules, `namespaceSelector` and `matchConditions` of the webhook.

Check the condition and the quotas of the namespaces:

```
d8 k get projectquotastatus XXX -o jsonpath='{range .status.conditions[*]}{.type}=
{.status} {.reason}: {.message}{"\n"}{end}'
d8 k get resourcequota -A -l projects.deckhouse.io/project=XXX -o custom-
columns=NAMESPACE:.metadata.namespace,HARD:.spec.hard,CALCULATED:.status.hard
```

- D8ProjectQuotaNotSupported CE S8
  ▼ The sum of a project quota is not enforced, because Deckhouse Platform does not manage the control plane.

  The cluster has projects with a quota, but the `control-plane-manager` module is disabled: Deckhouse Platform does not manage the control plane of the cluster. The sum over the namespaces of a project is enforced only where it does. Every ProjectQuotaStatus reports `Enforced=False` with the `ControlPlaneNotManaged` reason; the usage is counted and shown, and every namespace is limited by its `d8-project-quota` ResourceQuota, so no namespace alone uses more than the quota of the whole project.

  This is expected on a cluster whose control plane is managed outside Deckhouse Platform, for example by a cloud provider. Silence the alert if that is the case.

- D8ProjectQuotaOvershoot CE S5
  ▼ Usage of project XXX has grown past its quota.

  The usage of project `XXX` grew past its quota. The ledger allows that only for a status write of a quota saved long after it was approved (a change of the ledger leader at that moment, a stalled etcd, another slow webhook) or when kube-controller-manager recounts a quota, and then by at most a tenth of a limit. The `Exceeded` condition of the ProjectQuotaStatus names the keys, the difference and the probable cause. Nothing running is evicted; creating more of these resources is denied until the usage is back within the quota.

- D8ProjectQuotaSuspended CE S6
  ▼ The project quota ledger is suspended.

  The `quota-ledger-suspend` ConfigMap in `d8-multitenancy-manager` suspends the ledger: every write passes and the sum of the project quotas is not checked. Find who created it in the audit log and delete it when the reason is gone:

```
d8 k -n d8-multitenancy-manager delete configmap quota-ledger-suspend
```

- D8ProjectQuotaWebhookErrors CE S5
  ▼ The API servers fail to call the project quota ledger.

  For 10 minutes the API servers have been failing to call the webhook of the ledger: connections refused, timeouts or TLS errors. A change of the leader causes such errors for a second or two only. The steps of `D8ProjectQuotaLedgerDown` apply.

- D8ProjectQuotaWebhookMisconfigured CE S4
  ▼ The webhook of the project quota ledger is missing or misconfigured.

  The `d8-multitenancy-manager-quota-ledger` ValidatingWebhookConfiguration is missing, does not carry the CA of the `quota-ledger-certs` Secret, is changed so that writes of `d8-project-quota` do not reach the ledger (its rules, `namespaceSelector`, `objectSelector`, `matchConditions`, `failurePolicy` or service), or the API servers let the canary writes through without calling it. The sum of the project quotas is not checked; the `Enforced` condition of every ProjectQuotaStatus names the reason (`WebhookMissing`, `WebhookMisconfigured` or `WebhookNotCalled`) and the field or the API servers. Deckhouse puts the webhook back within about 4 minutes; if it does not, contact Deckhouse technical support. Compare the CA:

```
d8 k get validatingwebhookconfiguration d8-multitenancy-manager-quota-ledger -o
jsonpath='{.webhooks[0].clientConfig.caBundle}' | base64 -d | openssl x509 -noout -
subject
d8 k -n d8-multitenancy-manager get secret quota-ledger-certs -o jsonpath='{.data.ca\.crt}'
| base64 -d | openssl x509 -noout -subject
```

- D8ResourceQuotaAdmissionDegraded CE S4
  ▼ **The quota admission of the API servers is slow.**

  The 99th percentile of the ResourceQuota admission plugin is above one second. The plugin has a few workers for all creates in the cluster, so this slows down creates everywhere. The cause may be a slow webhook on `resourcequotas/status` (the project quota ledger or another one) or a slow etcd. Find the slow webhook by `apiserver admission webhook admission_duration_seconds` and check `etcd_disk_wal_fsync_duration_seconds`.

- MultitenancyManagerControllerUnavailable CE S4
  ▼ **The multitenancy-manager controller has had no available pods for more than 10 minutes.**

  The `multitenancy-manager` Deployment in the `d8-multitenancy-manager` namespace has had no available pods for more than 10 minutes.

  While this alert is active, projects are not reconciled, and the admission webhooks of the module, which refuse a request when they cannot be reached, refuse to create or change Project, ProjectTemplate and the other objects they check.

  Steps to troubleshoot:

  1. Check the pods of the controller:

     ```
     d8 k -n d8-multitenancy-manager get pods -l app=multitenancy-manager
     d8 k -n d8-multitenancy-manager describe pods -l app=multitenancy-manager
     ```

  2. A pod that is running but not ready may wait for the conversion of the CRDs. The controller does not start until each of the `clusterresourcegrantpolicies.multitenancy.deckhouse.io`, `projects.deckhouse.io` and `projecttemplates.deckhouse.io` CRDs converts through the webhook: until then, the API server would show it objects of an older version with only `apiVersion` rewritten. Look for the wait in the log:

     ```
     d8 k -n d8-multitenancy-manager logs deploy/multitenancy-manager -c multitenancy-manager |
     grep 'conversion'
     ```

     - `hold the controller until the webhook-handler of the deckhouse module registers the conversion webhook of the CRDs` means that the controller waits; the line names the CRDs and their current conversion strategy.
     - `the CRDs convert through the webhook` or `the CRDs already convert through the webhook` means that the wait is over.

  3. If the controller waits, check the conversion strategy of the CRDs and the ConversionWebhook resources of the module. The `webhook-handler` of the `deckhouse` module sets the strategy to `Webhook` for every ConversionWebhook resource:

     ```
     d8 k get crd clusterresourcegrantpolicies.multitenancy.deckhouse.io projects.deckhouse.io
     projecttemplates.deckhouse.io -o custom-
     columns=NAME:.metadata.name,STRATEGY:.spec.conversion.strategy
     d8 k get conversionwebhooks.deckhouse.io
     clusterresourcegrantpolicies.multitenancy.deckhouse.io projects.deckhouse.io
     projecttemplates.deckhouse.io
     d8 k -n d8-system logs deploy/webhook-handler -c handler --tail=200
     ```

     Do not set the strategy by hand: the conversion has to be served by the `webhook-handler`, which registers it once it has loaded the ConversionWebhook resource.

- MultitenancyManagerHelmProjectTemplate CE S9
  ▼ **ProjectTemplate XXX still renders its projects from a Helm text in resourcesTemplate.**

  ProjectTemplate `XXX` describes the objects of its projects with a Helm text in `resourcesTemplate`, and its projects are rendered from that text. They keep working as before. This form of a template is deprecated, and `resourcesTemplate` goes away in a future release, so move the template to structured fields and `manifests` before then. Moving a template from a Helm text describes the move step by step.

  The built-in templates `default`, `secure` and `secure-with-dedicated-nodes` are structured and accept the same parameters as their Helm texts did, except `administrators` and `resourceQuota`, so they show how a text maps to the fields. The full mapping is in Moving a template from a Helm text. Objects of the text that have no structured field go to `manifests`,

which the controller applies with the `d8:project:admin` role of the project. An AuthorizationRule of the `Admin` access level goes to `.spec.administrators` of each Project instead, because `manifests` does not create one. Cluster-scoped objects have no place in a template.

Once the template has structured fields in place of `resourcesTemplate`, the AuthorizationRule and ResourceQuota objects that `.spec.administrators` and `.spec.quota` replace are deleted after these fields work, and an AuthorizationRule that names nobody from `.spec.administrators` any more is deleted at once. Every other object of the text that the structured fields do not render under the same kind and name is left in place, and the `MultitenancyManagerProjectKeepsHelmTemplateObjects` alert names the project until `manifests` takes the object over or it is deleted. The administrators then have the `d8:project:admin` role in place of the `Admin` access level, and the two roles do not grant the same permissions. See Switching a project from a Helm text.

Steps:

1. Read the template and compare it with the built-in template it was copied from, if any:

   ```
   d8 k get projecttemplate XXX -o yaml
   d8 k get projecttemplate default -o yaml
   ```

2. List the projects of the template. A template that no project uses can be deleted, and the alert goes with it:

   ```
   d8 k get projects -l projects.deckhouse.io/project-template=XXX
   ```

3. Find the projects of the template that still set `administrators` or `resourceQuota` in `.spec.parameters`:

   ```
   d8 k get projects -l projects.deckhouse.io/project-template=XXX -o json | jq -r '.items[]
   | select((.spec.parameters // {}) | has("administrators") or
   has("resourceQuota")) | .metadata.name'
   ```

   These values get no standard fields and are not handed over, so move them to `.spec.administrators` and `.spec.quota` first, and only as far as they mean the same there. `.spec.administrators` gives users and groups the `d8:project:admin` role, so an administrator with another access level or another kind of subject needs a ProjectRoleBinding of its own. `.spec.quota` enforces every limit it holds, `0` included, so leave out a limit that the text did not render, such as a `0` that it skipped.

4. If the structured fields and `manifests` cover what the projects need, which they do unless the projects need cluster-scoped objects of the text or objects that the `d8:project:admin` role does not allow, write the template as `deckhouse.io/v1alpha2` with these fields and `manifests` and without `resourcesTemplate`, in one change, where the template is applied from, for example in Git or in CI. A template that projects use cannot lose `resourcesTemplate` without structured fields or `manifests` in its place, and once it has fields that `deckhouse.io/v1alpha1` cannot describe, a write of it through `deckhouse.io/v1alpha1` is refused. If the template is an unchanged copy of a built-in template, you can instead move its projects to that template in `.spec.projectTemplateName` and then delete the copy.

5. Check each project after the switch. Its `HandoverPending` condition is removed once the replaced objects are deleted, its `ManifestsApplied` condition says whether the objects of `manifests` are applied, and `.status.keptObjects` lists the objects left in place:

   ```
   d8 k get project PROJECT_NAME -o jsonpath='{range .status.conditions[*]}{.type}=
   {.status}: {.message}{"\n"}{end}{range .status.keptObjects[*]}{.kind}
   {.namespace}/{.name}{"\n"}{end}'
   ```

- MultitenancyManagerPreviousRevisionStillRunning CE S6
  ▾ Pods of several revisions of the multitenancy-manager controller have run side by side for more than 15 minutes.

  Pods of VALUE revisions of the `multitenancy-manager` Deployment in the `d8-multitenancy-manager` namespace have been pending or running side by side for more than 15 minutes.

  The rollout of the controller is stuck: the new pod is pending or not ready, or an old pod is stuck terminating on a lost node. The new pod waits for the pods of older revisions for 10 minutes at most and then starts its controllers anyway. If an old pod of a revision that runs without leader

election is still running, both pods write the Helm releases of the projects, and objects in the project namespaces may be deleted and created again.

Steps to troubleshoot:

1. List the controller pods with their revisions and nodes:

   ```
   d8 k -n d8-multitenancy-manager get pods -L pod-template-hash -o wide
   ```

2. Find out whether the new pod has started its controllers:

   ```
   d8 k -n d8-multitenancy-manager logs NEW_POD_NAME -c multitenancy-manager | grep 'other revisions'
   ```

   - `hold the controllers until the pods of other revisions finish` means that the new pod holds the lease and waits.
   - `no pods of other revisions to wait for` means that it has started its controllers at once, because the old pods had finished or were deleted more than 10 minutes ago.
   - `the pods of other revisions finished` means that it has started its controllers after waiting, because the old pods finished or were deleted more than 10 minutes ago.
   - `the pods of other revisions did not finish in time, starting the controllers anyway` means that it has started its controllers after 10 minutes of waiting.

   Without any of these lines, the new pod waits for the lease that the old pod holds. When the wait ends after 10 minutes, the new pod also writes a Warning event with the reason `PreviousRevisionStillRunning`, and the message of the event names the old pods. An event is kept for an hour by default:

   ```
   d8 k -n d8-multitenancy-manager get events --field-selector reason=PreviousRevisionStillRunning
   ```

3. Check the node of the old pod, shown in the `NODE` column:

   ```
   d8 k get node NODE_NAME
   ```

4. Act on what you see:

   - The old pod is `Running`, not terminating, and the new pod is not ready. The Deployment removes the old pods only after the new ones are ready, and deleting an old pod does not help, since its ReplicaSet creates it again. Find out why the new pod is not ready:

     ```
     d8 k -n d8-multitenancy-manager describe pod NEW_POD_NAME
     ```

   - The old pod is `Terminating` on a node that is `NotReady`. Such a pod stays until its node comes back or the Node object is deleted. If the node is lost for good, delete the Node object, and its pods are deleted with it. If you know the machine of the node is powered off, you can force-delete the old pod instead:

     ```
     d8 k -n d8-multitenancy-manager delete pod OLD_POD_NAME --force --grace-period=0
     ```

     Do not force-delete a pod whose containers may still run. The pod disappears from the API, while its process keeps writing the Helm releases.

- MultitenancyManagerProjectHandoverStuck CE S6
  ▼ Project XXX still keeps the AuthorizationRule and ResourceQuota objects its standard fields replace.

  An upgrade of the release of project `XXX` dropped AuthorizationRule or ResourceQuota objects that the Helm text of its template had rendered for the administrators and the quota of the project. These are the AuthorizationRule objects with the `Admin` access level of the administrators and a quota such as `all-pods` whose every limit `.spec.quota` sets. That happens when a template switches from `resourcesTemplate` to structured fields, when a project moves to such a template, or when the text no longer renders these objects. The controller kept them through the upgrade, so that the administrators never lost access and the namespace never lost its quota, and deletes them once the standard fields of the project replace them. The handover began more than 30 minutes ago and has not finished.

Until it does, the kept AuthorizationRule objects keep granting the `Admin` access level, the `user-authz:admin` role, next to the `d8:project:admin` role of the `d8-administrators` ProjectRoleBinding. The kept quotas keep applying next to the `d8-project-quota` ResourceQuota, and the tighter of them wins.

The kept objects are deleted by a reconcile of the project that succeeds once all of the following hold, and not earlier than 2 minutes after `d8-administrators` became `Ready` and after the handover began:

- the `d8-administrators` ProjectRoleBinding names the administrators from `.spec.administrators` and is `Ready` for its current generation, when a kept AuthorizationRule names one of them;
- the `d8:project:admin` ClusterRole has its aggregated rules, when a kept AuthorizationRule names one of the administrators;
- the `d8-project-quota` ResourceQuota has the limits from `.spec.quota` and is calculated, that is, its `status.hard` matches its `spec.hard`, when a kept quota has limits.

A kept AuthorizationRule that names nobody from `.spec.administrators` any more is deleted at once, and a subject removed from `.spec.administrators` is removed at once from a kept AuthorizationRule that names other administrators too. A reconcile that cannot find the template, validate the project against it, upgrade the release or apply the standard fields stops before the deletion, and the kept objects stay. A template that grants a role forbidden in projects puts the project in the `Error` state too, but it does not stop the deletion.

The `HandoverPending` condition of the project says what the controller waits for.

Steps to troubleshoot:

1. Read what the controller waits for, and the state and the other conditions of the project:

   ```
   d8 k get project XXX -o jsonpath='{.status.state}{"\n"}{range
   .status.conditions[*]}{.type}={.status}: {.message}{"\n"}{end}'
   ```

2. Check the administrators binding. Its `status.observedGeneration` must match `metadata.generation`, and its `Ready` condition must be `True`:

   ```
   d8 k -n XXX get projectrolebinding d8-administrators -o yaml
   ```

   If the binding lags behind, the controller could not create its RoleBinding in some namespace of the project. The logs of the controller name the namespace and the error:

   ```
   d8 k -n d8-multitenancy-manager logs -l app=multitenancy-manager -c multitenancy-manager --
   tail=-1 --prefix | grep d8-administrators
   ```

3. Check that the `d8:project:admin` ClusterRole has rules. The aggregation controller of kube-controller-manager fills them:

   ```
   d8 k get clusterrole d8:project:admin -o jsonpath='{.rules}'
   ```

4. Check the project quota and the events in the namespace of the project:

   ```
   d8 k -n XXX get resourcequota d8-project-quota -o yaml
   d8 k -n XXX get events --sort-by=.lastTimestamp
   ```

   A quota whose `status.hard` stays empty is not processed by the resource quota controller of kube-controller-manager.

5. Fix what the checks show, for example a RoleBinding that an admission policy refuses in one of the namespaces of the project, a kube-controller-manager that does not process the quota, or the condition that stops the reconcile of the project. The controller checks again whenever `d8-administrators` or `d8-project-quota` changes, and at least every minute, and deletes the kept objects once the conditions hold.

- MultitenancyManagerProjectKeepsHelmTemplateObjects CE S7
  ▼ Project XXX has VALUE objects its release left in place.

  The release of project `XXX` no longer holds VALUE objects, and the controller left them in place instead of deleting them. That happens when the project switches off a Helm text in `resourcesTemplate` and the structured fields do not render the object. It also happens when

`.spec.quota` stops covering a quota kept for a handover. The release of the structured fields leaves an object in place too when `manifests` of the template render it, so that `manifests` take it over. Examples are AuthorizationRule or RoleBinding objects, quotas, network policies, workloads or their volumes. The objects left in place behave as follows:

- They keep working as before. The rules and the bindings keep granting access, the quotas keep applying, and the workloads keep running.
- The project no longer manages them. A change of the template or of the project does not change them. When the project is deleted, the ones in its namespaces go with the namespaces, and the controller deletes the cluster-scoped ones that still carry the label.
- They carry the `projects.deckhouse.io/kept-from-helm-template: "true"` label and the `helm.sh/resource-policy: keep` annotation, and they have lost the `heritage: multitenancy-manager` label. While they carry the label, only a requester who may make the same change to objects of their kind across the cluster may change or delete them, so the users of the project may not.

The project lists them in `.status.keptObjects` until each of them is deleted, loses the label, is rendered by the template again, or is taken over by the `manifests` of the template. Decide for each object:

- If it is still needed, describe it in the template or in the project, for example with a ProjectRoleBinding in place of an AuthorizationRule or a RoleBinding, or with a limit in `.spec.quota`, and then delete the object. Or remove the label to hand it over to the project, whose users may then change and delete it.
- If it is in a namespace of the project, `manifests` of the template can take it over. Once `manifests` renders an object of the same kind, namespace and name, the controller applies it as the `system:multitenancy-manager:project:XXX` user, removes the label and takes the object off the list. The object then has what `manifests` renders, and a field that the release set and `manifests` leaves out is removed.
- If it is not needed, delete it. An access it grants ends only when it is deleted.
- If the template renders an object of the same kind and name again, the project release takes it over. The controller then removes the label, puts the `heritage: multitenancy-manager` label back and removes the `helm.sh/resource-policy: keep` annotation, unless the template sets it itself.

Steps:

1. List the objects:

   ```
   d8 k get project XXX -o jsonpath='{range .status.keptObjects[*]}{.kind}
   {.namespace}/{.name}{"\n"}{end}'
   ```

2. Keep an object as your own:

   ```
   d8 k -n NAMESPACE label KIND NAME projects.deckhouse.io/kept-from-helm-template-
   ```

3. Delete an object:

   ```
   d8 k -n NAMESPACE delete KIND NAME
   ```

The controller looks at the list every 10 minutes, and at a reconcile of the project at most once a minute.

- MultitenancyManagerProjectNamedLikeAdditionalNamespace CE S8
  ▼ Project XXX is named like an additional namespace of project XXX.

  The name of project `XXX` is the name of project `XXX`, a dash and a suffix. Additional namespaces of a project are named this way (`project-spec.name` of a ProjectNamespace), but `XXX` is not one of them. It is a separate project with a namespace of its own. The administrators, quota, bindings and template of project `XXX` do not apply to it.

  Such a pair appears when both namespaces existed before they became projects, or when a project is created under such a name. Nothing is broken, but the names read as one project to anyone who looks at them.

  If the two projects are meant to be separate, nothing has to change in the cluster. The alert keeps firing for as long as both projects exist, so silence it for this pair in Alertmanager.

  If `XXX` is meant to be an additional namespace of project `XXX`, recreate it as one. A namespace cannot move between projects in place. Deleting the project deletes its namespace with everything in it.

1. Make sure that everything running in the `XXX` namespace can be deployed again, or move it elsewhere first.
2. Find the ClusterProjectRoleBinding objects that name the ServiceAccounts of the `XXX` namespace. While a cluster-wide binding names them, the ProjectNamespace in step 4 is refused, so remove these subjects from the bindings first and add them back once the namespace is there again:

   ```
   d8 k get clusterprojectrolebindings -o json | jq -r --arg ns XXX '.items[] |
   select(any(.spec.subjects[]?; (.kind == "ServiceAccount" and .namespace ==
   $ns) or (.kind == "User" and (.name | startswith("system:serviceaccount:" +
   $ns + ":"))) or (.kind == "Group" and .name == "system:serviceaccounts:" +
   $ns))) | .metadata.name'
   ```

3. Delete the project. The command returns once the namespace is gone:

   ```
   d8 k delete project XXX
   ```

4. Create a ProjectNamespace in the `XXX` namespace, with `spec.name` set to the part of the name after `XXX-`. The controller creates the `XXX` namespace again, as an additional namespace of project `XXX`.
5. Deploy the workloads into the namespace.

Module node-local-dns

- D8NodeLocalDNSCacheDenialEvictionsHigh BE S6
  ▼ **node-local-dns denial cache is being frequently evicted.**

  The `denial` (NXDOMAIN) cache in node-local-dns is evicting entries frequently.

  This might indicate that there are Services that are being recreated too often in cluster. So, denial cache size is not enough (`denial 65536` is currently configured).

  Affected node: `NODE_NAME`

- D8NodeLocalDNSCacheSuccessEvictionsHigh BE S6
  ▼ **node-local-dns success cache is being frequently evicted.**

  The `success` cache in node-local-dns is evicting entries frequently (>100 in 5 minutes).

  This could mean there are too many different DNS records application have to resolve (to many Services?). So, node-local-dns cache capacity is too small (`success 120000` is currently configured), or traffic patterns changed.

  Affected node: `NODE_NAME`

- D8NodeLocalDNSForwardMaxConcurrentRejects BE S5
  ▼ **The forward plugin is rejecting DNS queries due to max concurrency limit.**

  More than 20 queries were rejected within the last 5 minutes on node `NODE_NAME` because the max concurrent query limit was reached.

- D8NodeLocalDNSKubeforwardRequestLatencyP95High BE S4
  ▼ **High DNS request latency in kubeforward (p95 > 300ms under load).**

  The 95th percentile DNS request latency for A-records through kubeforward exceeded 300 milliseconds over the last 5 minutes while handling more than 0.1 QPS.

  This may indicate slow upstreams, network issues, or DNS overload on this node.

  Affected node: `NODE_NAME`

  For more detailed diagnostics, inspect `Top 5 upstreams by slow kubeforward requests` in Grafana.

  If the alert fires only on one node, run the checks below from node `NODE_NAME` (for example, from a debug pod on this node). If the alert fires on many nodes, treat it as a potential systemic issue first.

  Get kubeforward upstream IPs from a master host:

  ```
  d8 k -n kube-system get po -l k8s-app=kube-dns -o jsonpath='{.items[*].status.podIP}'
  ```

Then run latency checks on the affected node `NODE_NAME` (or from a debug pod on that node):

```
UPSTREAMS="paste kube-dns pod IPs here"
for ip in $UPSTREAMS; do
  echo "== upstream ${ip} =="
  for i in $(seq 1 20); do
    dig @"$ip" -p 5353 kubernetes.default.svc.cluster.local A +tries=1 +time=1 +stats | grep
"Query time"
  done
done
```

To identify slow requests, enable module logs (`enableLogs`) and inspect slow query messages:

```
pod=$(kubectl -n kube-system get po -l k8s-app=node-local-dns --field-selector
spec.nodeName=NODE_NAME -o jsonpath='{.items[0].metadata.name}')
d8 k -n kube-system logs "$pod" | grep '\[kubeforward\] slow query'
```

Example log line: `[kubeforward] slow query A kubernetes.default.svc.cluster.local. took 425ms (rcode=NOERROR, upstream=10.96.0.10:53)`

Interpretation:

- If most upstreams are slow and many nodes are affected, suspect systemic network/CoreDNS overload.
- If one upstream is slow or only one node is affected, suspect a localized issue.

- D8NodeLocalDNSUpstreamHealthcheckFailed BE S5
  ▼ **Too many upstream health check failures in forward plugin.**

  The health check system of the `forward` plugin failed more than 10 times within 5 minutes on node `NODE_NAME`. As a result, all upstreams were considered unhealthy.

Module node-manager

- CapsInstanceUnavailable CE S8
  ▼ **There are unavailable instances in the MACHINE_DEPLOYMENT_NAME MachineDeployment.**

  The MachineDeployment `MACHINE_DEPLOYMENT_NAME` has **VALUE** unavailable instances.

  Check the status of the instances in the cluster with the following command:

  ```
  d8 k get instance -l node.deckhouse.io/group=MACHINE_DEPLOYMENT_NAME
  ```

- ClusterHasOrphanedDisks CE S6
  ▼ **Cloud data discoverer found orphaned disks in the cloud.**

  The cloud data discoverer has found disks in the cloud that do not have a corresponding PersistentVolume in the cluster.

  You can safely delete these disks manually from your cloud provider:

  ID: ID, Name: NAME

- D8BashibleApiserverLocked CE S6
  ▼ **Bashible-apiserver has been locked for too long.**

  `Bashible-apiserver` has been locked for an extended period.

  To resolve the issue, check if the `bashible-apiserver` Pods are up-to-date and running:

  ```
  d8 k -n d8-cloud-instance-manager get pods -l app=bashible-apiserver
  ```

- D8CAPIControllerManagerNotAvailable CE S4
  ▼ **No capi-controller-manager replica is available.**

  No replica of `capi-controller-manager` has been available for more than 5 minutes.

  Its admission webhooks reject changes to Cluster API objects, so node groups served by Cluster API do not create or remove nodes.

To inspect the pods:

```
d8 k -n d8-cloud-instance-manager get pods -l app=capi-controller-manager -o wide
```

- D8CAPIControllerManagerReplicasUnavailable CE S7
  ▼ **Part of the capi-controller-manager replicas is unavailable.**

  Some replicas of `capi-controller-manager` have been unavailable for more than 30 minutes.

  The controller keeps working while at least one replica is available. Usually a master is down, cordoned, or its port is taken.

  To inspect the pods:

  ```
  d8 k -n d8-cloud-instance-manager get pods -l app=capi-controller-manager -o wide
  ```

- D8CloudDataDiscovererCloudRequestError CE S6
  ▼ **Cloud data discoverer cannot get data from the cloud.**

  Refer to the cloud data discoverer's logs for details:

  ```
  d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
  ```

- D8CloudDataDiscovererSaveError CE S6
  ▼ **Cloud data discoverer cannot save data to a Kubernetes resource.**

  Refer to the cloud data discoverer's logs for details:

  ```
  d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
  ```

- D8ClusterAutoscalerManagerPodIsNotReady CE S8
  ▼ **The POD_NAME Pod is NOT Ready.**

  To check the Pod's status, run the following command:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- D8ClusterAutoscalerPodIsNotRunning CE S8
  ▼ **The cluster-autoscaler Pod is NOT Running.**

  The POD_NAME Pod is STATUS.

  To check the Pod's status, run the following command:

  ```
  d8 k -n NAMESPACE get pods POD_NAME -o json | jq .status
  ```

- D8ClusterAutoscalerPodIsRestartingTooOften CE S9
  ▼ **Too many cluster-autoscaler restarts detected.**

  The cluster-autoscaler has restarted VALUE times in the past hour.

  Frequent restarts may indicate a problem. The cluster-autoscaler is expected to run continuously without interruption.

  Check the logs for details:

  ```
  d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -c cluster-autoscaler
  ```

- D8ClusterAutoscalerTargetAbsent CE S8
  ▼ **Cluster-autoscaler target is missing in Prometheus.**

  The cluster-autoscaler automatically scales nodes in the cluster. If it's unavailable, it will be impossible to add new nodes when there's not enough resources for scheduling Pods. It may also lead to unnecessary cloud costs due to unused but still provisioned cloud instances.

  To resolve the issue, follow these steps:

  1. Check availability and status of cluster-autoscaler Pods:

```
d8 k -n d8-cloud-instance-manager get pods -l 'app in (cluster-autoscaler,cluster-autoscaler-mcm)'
```

2. Verify that the cluster-autoscaler Deployment exists:

```
d8 k -n d8-cloud-instance-manager get deploy -l 'app in (cluster-autoscaler,cluster-autoscaler-mcm)'
```

3. Check the Deployment’s status:

```
d8 k -n d8-cloud-instance-manager describe deploy deployment-name
```

- D8ClusterAutoscalerTargetDown CE S8

  Prometheus is unable to scrape cluster-autoscaler’s metrics.

- D8ClusterAutoscalerTooManyErrors CE S8
  ▼ Cluster-autoscaler is issuing too many errors.

  The cluster-autoscaler encountered multiple errors from the cloud provider when attempting to scale the cluster.

  Check the logs for details:

  ```
  d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -c cluster-autoscaler
  ```

- D8FencingAgentDown CE S6
  ▼ The fencing agent does not work on node NODE_NAME.

  The node `NODE_NAME` is Ready, but Prometheus does not get the metrics of its fencing agent for 5 minutes.

  Without the agent, the node does not detect a loss of the control plane, and its neighbors do not see it.

  To check the pod of the agent:

  ```
  d8 k -n d8-cloud-instance-manager get pods -l app=fencing-agent --field-selector spec.nodeName=NODE_NAME -o wide
  d8 k -n d8-cloud-instance-manager describe pods -l app=fencing-agent --field-selector spec.nodeName=NODE_NAME
  ```

- D8FencingAgentInvalidKey CE S6
  ▼ The fencing agent on node NODE_NAME cannot read the memberlist key.

  The fencing agent on the node `NODE_NAME` cannot read the memberlist key from the Secret `fencing-memberlist-key` in the namespace `d8-cloud-instance-manager`, because the key has a wrong format. The agent does not connect to the other agents and does not arm the watchdog, so the node is not protected.

  The `primary` key must have exactly 32 bytes. A frequent error is a key written as a base64 string into `stringData`. To check the length of the key:

  ```
  d8 k -n d8-cloud-instance-manager get secret fencing-memberlist-key -o jsonpath='{.data.primary}' | base64 -d | wc -c
  ```

  The command must show 32. To write the key correctly, refer to the module FAQ.

- D8FencingAgentRestarted CE S7
  ▼ The fencing agent restarted on node NODE_NAME.

  The container `fencing-agent` restarted on the node `NODE_NAME` in the last 15 minutes.

  After a crash of the agent, the watchdog stays armed. At the start, the new agent process disarms the watchdog if the control plane is available. If the control plane is not available, the watchdog timeout expires and the kernel panics. The node then stays stopped, because `kernel.panic` is `0`. Restart the node manually.

  If the agent crashes again and again, the watchdog of the node is not armed, and fencing does not

stop the node if the node fails.

To read the logs of the agent that crashed:

```
d8 k -n d8-cloud-instance-manager logs -l app=fencing-agent --field-selector
spec.nodeName=NODE_NAME -c fencing-agent --previous
```

- D8FencingEvictionNotProgressing CE S4
  ▼ **The evacuation of node NODE_NAME is not complete after 60 seconds.**

  The fencing controller put the taint `node.kubernetes.io/out-of-service` on the node `NODE_NAME`, but the evacuation is not complete after 60 seconds. The `FencingFailedNodeState` object stays in the phase `Evicting` and has the condition `Progressing` with the status `False` and the reason `XXX`. The controller examines the node again every 5 seconds. When the cause is gone, the evacuation completes, and the alert stops.

  The reasons:

  - `VolumesAttached`: the node still has `VolumeAttachment` objects. The volumes of its pods are not free for other nodes. Usually the CSI driver or the storage cannot detach the volumes.
  - `PodsRemaining`: the controller could not delete pods of the node, or another error of the API stops the evacuation. The message gives the pods and the errors.
  - `TooManyUnhealthyMachines`: the controller does not delete the node of a cloud machine, because the MachineHealthCheck does not replace it. More than 50 percent of the machines are unhealthy, or the cluster has no MachineHealthCheck that reports its machines. The message gives the numbers.

  To read the message of the condition:

  ```
  d8 k get fencingfailednodestate NODE_NAME -o yaml
  ```

  To find the `VolumeAttachment` objects of the node:

  ```
  d8 k get volumeattachments -o custom-
  columns=NAME:.metadata.name,NODE:.spec.nodeName,ATTACHER:.spec.attacher,PV:.spec.source.persistentVolu
  meName,ATTACHED:.status.attached | awk 'NR==1 || $2=="NODE_NAME"'
  ```

  To read the errors of the detach:

  ```
  d8 k get volumeattachments -o json | jq '.items[] | select(.spec.nodeName=="NODE_NAME")
  | {name: .metadata.name, detachError: .status.detachError}'
  ```

- D8FencingKernelPanicMismatch CE S6
  ▼ **The value of kernel.panic on node NODE_NAME is not the value that fencing needs.**

  The value of `kernel.panic` on the node `NODE_NAME` stays wrong for 30 minutes. The fencing agent does not arm the watchdog, so the node is not protected.

  Fencing needs `kernel.panic` set to `0`. After a fencing panic, the node must stay stopped and must not restart by itself.

  Bashible sets the value. Check that bashible works on the node and that no other tool changes `kernel.panic`:

  ```
  d8 k get node NODE_NAME -o jsonpath='{.status.conditions[?(@.type=="Fencing")]}' | jq
  sysctl kernel.panic
  ```

- D8FencingKeyRegenerated CE S9
  ▼ **The memberlist key of fencing was generated again.**

  Deckhouse generated a new memberlist key of fencing, because the Secret `fencing-memberlist-key` in the namespace `d8-cloud-instance-manager` was deleted. The agents use the new key. The old key is lost.

  Agents that did not get the new key cannot talk with the other agents. If you keep a copy of the key outside the cluster, replace it.

  The alert goes out 24 hours after the key was generated. It does not rise when Deckhouse creates the key for the first time or when you rotate the key yourself.

- D8FencingNodeAliveAfterAttestation CE S4

▾ Node NODE_NAME sends heartbeats after a witness confirmed its stop.

A witness node confirmed that the node `NODE_NAME` stopped. After that, the fencing controller saw a heartbeat of the node. Thus the node can still work.

The fencing controller does not put the taint `node.kubernetes.io/out-of-service` on the node while the node sends heartbeats. When the heartbeats stop, the node gets the taint after the silent delay: `controlPlaneLostAfter` + 0.2s + `watchdog.timeout` + `watchdog.margin` + 0.9s of the profile after the last sign of life of the node.

Examine the node. To read the state of the incident:

```
d8 k get fencingfailednodestate NODE_NAME -o yaml
```

- D8FencingNodeFailed CE S4
  ▾ Fencing found a failure of node NODE_NAME.

  The node `NODE_NAME` has a `FencingFailedNodeState` object. The fencing agents of other nodes found that this node failed. The watchdog of the node is armed.

  The fencing controller puts the taint `node.kubernetes.io/out-of-service` on the node when the node is stopped, for one of these reasons:

  - A witness node confirmed that the node stopped, and the watchdog timeout of the node passed after the confirmation.
  - The node is silent: it does not send its heartbeat through the API server. A node that loses the control plane stops itself. The controller waits for the time that the node needs to stop: the control plane loss threshold, the watchdog timeout and a margin after the last sign of life.

  After the taint, the pods of the node move to other nodes. The phase of the object shows the progress. For the cases when the controller does not put the taint, refer to the FAQ of the `node-manager` module.

  If the phase is `FallbackAlive`, the node works. Refer to the alert `D8FencingNodeIsolatedFromPeers`.

  To read the state of the incident:

  ```
  d8 k get fencingfailednodestate NODE_NAME
  d8 k describe fencingfailednodestate NODE_NAME
  ```

  The alert stops when the object is deleted.

- D8FencingNodeIPMismatch CE S6
  ▾ The fencing agent on node NODE_NAME cannot find the IP address of the node.

  The fencing agent on the node `NODE_NAME` cannot use the IP address of its Node object. The Node object has no IPv4 address, or the node network interface does not have this address. The agent waits for the address and does not arm the watchdog, so the node is not protected.

  To compare the addresses:

  ```
  d8 k get node NODE_NAME -o jsonpath='{.status.addresses}' | jq
  ```

- D8FencingNodeIsolatedFromPeers CE S5
  ▾ Node NODE_NAME works, but the fencing agents of other nodes cannot connect to it.

  The fencing agents of other nodes found that the node `NODE_NAME` failed. But the node connects to the control plane and sends its heartbeat. Thus the node works, and the network between this node and the other nodes is broken.

  Usually a firewall rule or an ACL stops the traffic of the fencing agents. The fencing controller does not put the taint `node.kubernetes.io/out-of-service` on a node that sends its heartbeat.

  To read the state of the incident:

  ```
  d8 k get fencingfailednodestate NODE_NAME -o yaml
  ```

  Make sure that the network lets the fencing agents connect to each other.

- D8FencingProfileIsUnavailable CE S5
  ▾ Fencing of node NODE_NAME is blocked, because the controller cannot use its SLA profile.

The node `NODE_NAME` has a fencing incident, but the fencing controller cannot use the SLA profileXXX `XXX` XXX of the incident. The controller does not put the taint `node.kubernetes.io/out-of-service` on the node, and the pods of the node do not move to other nodes.

The controller calculates the times of the incident from the profile:

- the time from the confirmation of a witness to the taint: `watchdog.timeout` + `watchdog.margin`;
- the time from the last sign of life of a silent node to the taint: `controlPlaneLostAfter`, `watchdog.timeout`, `watchdog.margin` and a fixed margin;
- the time after the last heartbeat of the node: `heartbeat.interval` and `heartbeat.apiTimeout`.

The message of the condition `ConfigurationError` of the incident gives the cause, for example the profile that is missing.

The profile of the incident is the profile of the node group of the node. Only an armed node needs a profile. For a node that is not armed, the controller uses a fixed time of 20 seconds after the last heartbeat and reads no profile.

The controller tries again with backoff. Thus the alert shows that the profile is not usable for some minutes.

The module installs the built-in profiles. If a profile is missing, a user usually deleted or changed it. XXX To read the profile:

```
d8 k get fencingslaprofile XXX -o yaml
```

XXX The controller did not find the name of the profile. The message of the condition `ConfigurationError` gives the cause. XXX

To read the cause and the events of the incident:

```
d8 k get fencingfailednodestate NODE_NAME -o jsonpath='{.status.conditions}' | jq
d8 k describe fencingfailednodestate NODE_NAME
```

To read the logs of the controller:

```
d8 k -n d8-cloud-instance-manager logs deploy/fencing-controller -c fencing-controller --all-pods=true
```

- D8FencingSystemNetworkNotFound CE S4
  ▼ **The SystemNetwork XXX of fencing does not exist.**

  The ModuleConfig `node-manager` names the SystemNetwork `XXX` in `settings.fencing.systemNetworkFallback.systemNetworkName`, but the cluster has no such SystemNetwork. The agents cannot use the second network.

  Deckhouse does not change the setting. Create the SystemNetwork, or set the name of another one, or delete the name to turn the second network off.

  To check the network:

  ```
  d8 k get systemnetwork XXX
  d8 k get moduleconfig node-manager -o jsonpath='{.spec.settings.fencing}'
  ```

  The alert goes out about 10 minutes after the network is back, because it looks at the last 10 minutes.

  While all node groups have the mode `Disabled`, the rules of fencing are not loaded, so the alert does not rise. The agents do not use the second network then. The alert rises when you turn fencing on.

- D8FencingWatchdogDeviceMissing CE S6
  ▼ **The fencing agent on node NODE_NAME has no watchdog device.**

  The fencing agent on the node `NODE_NAME` has no watchdog device for 30 minutes. The agent does not arm the watchdog, so the node is not protected.

  A new agent pod does not start without `/dev/watchdog`. Thus, usually this agent runs in a pod of the previous revision without the watchdog device, and the DaemonSet did not replace this pod

yet. On the node, `softdog` and `/dev/watchdog` can be present.

Do these checks:

1. Check the rollout of the DaemonSet of the node group. Find the agent pods of the group in the states `ContainerCreating` (with the `FailedMount` event) or `Terminating` on other nodes. Such a pod stops the rollout. A pod does not start if the NodeGroupConfiguration step failed on its node. A pod stays `Terminating` if its node is `NotReady`.

   ```
   d8 k -n d8-cloud-instance-manager rollout status ds/fencing-agent-group
   d8 k -n d8-cloud-instance-manager get pods -l app=fencing-agent,node-group-name=group -o wide
   d8 k -n d8-cloud-instance-manager get events --field-selector reason=FailedMount
   ```

2. On the node, check that bashible ran the NodeGroupConfiguration `enable-watchdog-for-nodegroup-*` without errors, that `softdog` is loaded and that `/dev/watchdog` is present:

   ```
   d8 k get node NODE_NAME -o jsonpath='{.status.conditions[?(@.type=="Fencing")]}' | jq
   journalctl -u bashible --since -1h
   lsmod | grep softdog
   ls -l /dev/watchdog
   ```

- D8FencingWatchdogTimeoutMismatch CE S6
  ▼ The watchdog timeout on node NODE_NAME differs from the timeout of the fencing profile.

  The watchdog of the node `NODE_NAME` reports a timeout that is not the timeout of the fencing profile of its node group. The fencing agent does not arm the watchdog, so the node is not protected.

  A watchdog driver can round the timeout or accept only some values. To read the reason:

  ```
  d8 k get node NODE_NAME -o jsonpath='{.status.conditions[?(@.type=="Fencing")]}' | jq
  ```

- D8KubeletCheckpointStateUnpinnedContainers CE S4
  ▼ Pod NAMESPACE/POD_NAME may run in a NUMA zone other than the one recorded by kubelet.

  Kubelet on Node `NODE NAME` reset the memory manager checkpoint and recorded new NUMA zones for Pod `NAMESPACE/POD_NAME`. Zones are applied only when a container is created, so the running containers stay in their previous zones, which may differ from the recorded ones.

  As a result, the zone the Pod really uses may also be assigned to another Guaranteed Pod, causing both Pods to share its memory resources. The Pod's CPUs and memory may also be assigned to different NUMA zones, which slows down its memory access.

  The Pod is not restarted automatically. The restart time depends on the workload. To restore resource isolation, do the following:

  - To recreate the Pod's containers in newly assigned NUMA zones, delete the Pod:

    ```
    d8 k -n NAMESPACE delete pod POD_NAME
    ```

  - For a virtual machine, perform a live migration to another node instead, without stopping it.
  - If the node runs many such Pods, you can drain it instead of deleting them one by one.

  The alert resolves within a few minutes after the Pod is recreated. No kubelet restart is needed.

- D8MachineControllerManagerPodIsNotReady CE S8

  The POD_NAME Pod is NOT Ready.

- D8MachineControllerManagerPodIsNotRunning CE S8
  ▼ The machine-controller-manager Pod is NOT Running.

  The POD_NAME Pod is STATUS.

  To check the Pod's status, run the following command:

  ```
  d8 k -n NAMESPACE get pods POD_NAME -o json | jq .status
  ```

- D8MachineControllerManagerPodIsRestartingTooOften CE S9
  ▼ Too many machine-controller-manager restarts detected.

  The `machine-controller-manager` has restarted VALUE times in the past hour.

  Frequent restarts may indicate a problem. The `machine-controller-manager` is expected to run continuously without interruption.

  Check the logs for details:

  ```
  d8 k -n d8-cloud-instance-manager logs -f -l app=machine-controller-manager -c controller
  ```

- D8MachineControllerManagerTargetAbsent CE S8
  ▼ Machine-controller-manager target is missing in Prometheus.

  `Machine-controller-manager` controls ephemeral nodes in the cluster. If it becomes unavailable, it will be impossible to create or delete nodes in the cluster.

  To resolve the issue, follow these steps:

  1. Check availability and status of `machine-controller-manager` Pods:

     ```
     d8 k -n d8-cloud-instance-manager get pods -l app=machine-controller-manager
     ```

  2. Verify availability of the `machine-controller-manager` Deployment:

     ```
     d8 k -n d8-cloud-instance-manager get deployment machine-controller-manager
     ```

  3. Check the Deployment’s status:

     ```
     d8 k -n d8-cloud-instance-manager describe deployment machine-controller-manager
     ```

- D8MachineControllerManagerTargetDown CE S8

  Prometheus is unable to scrape the machine-controller-manager’s metrics.

- D8NeedMigrateStateToOpenTofu CE S4
  ▼ Terraform state must be migrated to OpenTofu.

  This likely means the automatic migration to OpenTofu was canceled due to destructive changes detected in the cluster state.

  Verify the following:

  - Current Terraform version:

    ```
    d8 k -n d8-system exec -it deployments/terraform-state-exporter -c exporter -- terraform version
    ```

  - Version in the Terraform state:

    ```
    d8 k -n d8-system get secret d8-cluster-terraform-state -o json | jq -r '.data["cluster-tf-state.json"] | @base64d | fromjson | .terraform_version'
    ```

  - Check for destructive changes:

    ```
    d8 k exec -it deploy/terraform-state-exporter -n d8-system -- dhctl terraform check
    ```

  To resolve the issue and migrate to OpenTofu manually, follow these steps:

  1. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run:

     ```
     dhctl converge
     ```

     This should resolve all destructive changes in the cluster.

  2. After that, the `terraform-auto-converger` Pod should complete the migration

automatically.

If the automatic migration doesn't happen, use the `install` container with the current Deckhouse release and run the following command to enforce migration of the Terraform state to OpenTofu:

```
dhctl converge-migration
```

If this alert appears in a cluster created by Deckhouse Commander, it means the cluster has been updated to a new Deckhouse version without approving state changes (for example, destructive changes).

To resolve this, follow these steps:

1. In Deckhouse Commander, click **Delete manually** to temporarily remove the cluster from control.
2. In the cluster, remove the `commander-agent` module using the following command:

   ```
   d8 k delete mc commander-agent
   ```

3. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run the following command to resolve all destructive changes in the cluster:

   ```
   dhctl converge
   ```

4. In Deckhouse Commander, click **Attach** to reattach the cluster.

- D8NodeCgroupV2NotSupported CE S8
  ▼ **Node NODE_NAME is using deprecated cgroup v1 – migration to v2 required.**

  Node NODE_NAME in NodeGroup NODE_GROUP_NAME does not support cgroup v2. Starting from Kubernetes 1.35, cgroup v1 is deprecated. To migrate to cgroup v2, please upgrade the kernel to version 5.8 or newer. For more information, please refer to the documentation.

- D8NodeContainerdV2NotSupported CE S8
  ▼ **Node NODE_NAME is using deprecated containerd v1 – migration to v2 required.**

  Node NODE_NAME in NodeGroup NODE_GROUP_NAME does not support containerd v2. Starting from Kubernetes 1.38, containerd v1.y will no longer be supported. Please check the requirements for containerd v2 and schedule your migration plan.

  The exact reason is listed in the `node.deckhouse.io/containerd-v2-err` annotation on the node.

  If the reason is `erofs` and the node runs Ubuntu in AWS, the module is shipped in a separate `linux-modules-extra` package. Deckhouse Platform installs it automatically, but for some kernels, mostly outdated ones, the package is no longer published in the APT repository. In that case, the installation is skipped and a warning is written to the node logs. The problem has been confirmed for the `5.15.0-1028-aws`, `6.8.0-1024-aws`, and `6.8.0-1029-aws` kernels. Switch the node group to a newer AMI or kernel version that includes `erofs`.

- D8NodeControllerReconcileErrorsHigh CE S8
  ▼ **Node-controller reconcile errors are high for CONTROLLER_NAME.**

  Controller `CONTROLLER_NAME` has a reconcile error rate above 0.1 errors per second for more than 10 minutes.

  Check node-controller logs:

  ```
  kubectl -n d8-cloud-instance-manager logs deploy/node-controller --all-pods=true
  ```

- D8NodeControllerReconcileP99High CE S8
  ▼ **Node-controller reconcile p99 is high for CONTROLLER_NAME.**

  The p99 reconcile duration for controller `CONTROLLER_NAME` has been above 5 seconds for more than 10 minutes.

  This may indicate slow reconciliation, Kubernetes API latency, or degraded controller performance.

  To inspect node-controller logs:

```
kubectl -n d8-cloud-instance-manager logs deploy/node-controller --all-pods=true
```

- D8NodeControllerWorkqueueDepthHigh CE S8
  ▼ **Node-controller workqueue NAME has a high backlog.**

  The node-controller workqueue `NAME` has more than 50 queued objects for more than 15 minutes.

  This may indicate that the controller cannot process reconciliation events fast enough.

  To inspect node-controller logs:

  ```
  kubectl -n d8-cloud-instance-manager logs deploy/node-controller --all-pods=true
  ```

- D8NodeGroupFencingDeprecatedSettings CE S9
  ▼ **NodeGroup NAME uses a deprecated fencing setting.**

  NodeGroup `NAME` has a deprecated fencing setting in `spec.fencing`:

  - `mode: Watchdog` works as `mode: Slow`. To remove the alert, set `mode: Slow`, or choose another mode.
  - `watchdog.timeout` has no effect. The timeout comes from the fencing profile of the mode. To remove the alert, delete `watchdog.timeout`.

  To read the settings of the NodeGroup:

  ```
  d8 k get nodegroup NAME -o jsonpath='{.spec.fencing}'
  ```

- D8NodeGroupIsNotUpdating CE S8
  ▼ **NodeGroup NODE_GROUP_NAME is not handling the update correctly.**

  There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. Although Nodes have detected the update, none of them have received approval to start the update process.

  Most likely, there is a problem with the `update_approval` hook of the `node-manager` module.

- D8NodeIsNotUpdating CE S7
  ▼ **Node NODE_NAME cannot complete the update.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received approval, started the update, and encountered a step that could cause downtime. The update manager (the `update_approval` hook of the `node-group` module) granted downtime approval, but no success message was received, which indicates that the update has not completed.

  To investigate the details, view Bashible logs on the Node:

  ```
  journalctl -fu bashible
  ```

- D8NodeIsNotUpdating CE S8
  ▼ **Node NODE_NAME cannot complete the update.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, requested and received approval, but failed to complete the update.

  To investigate the details, view Bashible logs on the Node:

  ```
  journalctl -fu bashible
  ```

- D8NodeIsNotUpdating CE S9
  ▼ **Node NODE_NAME is not updating.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has a pending update but it’s neither receiving it nor attempting to.

  Most likely, Bashible is not handling the update correctly. It should annotate the Node with `update.node.deckhouse.io/waiting-for-approval` before the update can proceed.

  Options to investigate the details:

  - Check the expected configuration checksum for the NodeGroup:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o jsonpath=
{.data.NODE_GROUP_NAME} | base64 -d
```

  - Check the current configuration checksum on the Node:

```
d8 k get node NODE_NAME -o
jsonpath='{.metadata.annotations.node\.deckhouse\.io/configuration-
checksum}'
```

  - View Bashible logs on the Node:

```
journalctl -fu bashible
```

- D8NodeIsUnmanaged CE S9
  ▼ The NODE_NAME Node is not managed by the node-manager module.

  The NODE_NAME Node is not managed by the `node-manager` module.

  To resolve this issue, follow the instructions how to clean up the node and add it to the cluster.

- D8NodeOutOfServiceTainted CE S6
  ▼ Node NODE_NAME has the taint node.kubernetes.io/out-of-service for 30 minutes.

  The node `NODE_NAME` has the taint `node.kubernetes.io/out-of-service` for 30 minutes. Kubernetes does not run pods on the node.

  The fencing controller or a user puts the taint on a node that stopped. Start the node again or remove it from the cluster.

  To read the taints of the node:

```
d8 k get node NODE_NAME -o jsonpath='{.spec.taints}' | jq
```

  If a user put the taint and the node works, remove the taint:

```
d8 k taint node NODE_NAME node.kubernetes.io/out-of-service-
```

- D8NodeUpdateStuckWaitingForDisruptionApproval CE S8
  ▼ Node NODE_NAME cannot obtain disruption approval.

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received initial approval, and started the update. However, it reached a stage that could cause downtime and was unable to obtain disruption approval. A disruption approval is normally issued automatically by the `update_approval` hook of the `node-manager` module.

  To resolve this issue, investigate why the approval could not be granted to proceed with the update.

- D8ProblematicNodeGroupConfiguration CE S8
  ▼ Node NODE_NAME cannot begin the update.

  There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. However, Node `NODE_NAME` cannot begin the update.

  The Node is missing the `node.deckhouse.io/configuration-checksum` annotation, which may indicate that its bootstrap process did not complete correctly.

  Troubleshooting options:

  - Check the `cloud-init` log (`/var/log/cloud-init-output.log`) on the node.
  - Check the NodeGroupConfiguration resource associated with the `NODE_GROUP_NAME` NodeGroup for potential issues.
- NodeGroupGPUModuleDisabled CE S6
  ▼ NodeGroup NAME has spec.gpu, but the gpu module is disabled.

  NodeGroup `NAME` has `spec.gpu` configured, but the `gpu` module is disabled.

  Built-in GPU support has been removed from the `node-manager` module. GPU nodes in this NodeGroup are not prepared and do not publish GPU resources, so Pods requesting `nvidia.com/gpu` are not scheduled on them.

To resolve this issue, do one of the following:

- Enable the `gpu` module (available in the Core and Ultimate/EE editions).
- If the NodeGroup does not need GPU, remove the `spec.gpu` section from it.

- NodeGroupHasStaticInternalNetworkCIDRsField CE S9
  ▼ **NodeGroup NAME uses deprecated field spec.static.internalNetworkCIDRs.**

  The `spec.static.internalNetworkCIDRs` field is deprecated and has been moved to the static cluster configuration.

  To resolve this issue, delete the field from NodeGroup NAME. This is safe, as the setting has already been migrated automatically.

- NodeGroupMasterTaintIsAbsent CE S4
  ▼ **The master NodeGroup is missing the required control-plane taint.**

  The `master` NodeGroup doesn't have the `node-role.kubernetes.io/control-plane: NoSchedule` taint. This may indicate a misconfiguration where control-plane nodes can run non-control-plane Pods.

  To resolve the issue, add the following to the `master` NodeGroup spec:

  ```
  nodeTemplate:
    taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/control-plane
  ```

  Note that the taint `key: node-role.kubernetes.io/master` is deprecated and has no effect starting from Kubernetes 1.24.

- NodeGroupProviderTypeIsInvalid CE S4
  ▼ **The spec.providerType field is set incorrectly in NodeGroup NAME.**

  The `spec.providerType` field of NodeGroup `NAME` is set to `XXX`, which is not the cloud provider the nodes of the group run in.

  To resolve this issue, set the field to a correct value:

  - If the nodes of the group run in a cloud, set the field to the `type` of that cloud provider, as published in its registration secret:

    ```
    kubectl -n kube-system get secrets -l cloud-provider.deckhouse.io/registration -o json \
    | jq -r '.items[] | .data.type | @base64d'
    ```

  - If the cluster is static, or the NodeGroup has `nodeType: Static`, set the field to `None` or leave it empty:

    ```
    kubectl patch nodegroup NAME --type=merge -p '{"spec":{"providerType":"None"}}'
    ```

- NodeGroupReplicasUnavailable CE S7
  ▼ **NodeGroup NODE_GROUP_NAME has no available instances.**

  This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

  1. Cloud provider resource limits.
  2. No access to the cloud provider API.
  3. Misconfiguration of the cloud provider or instance class.
  4. Problems with bootstrapping the Machine.

  Recommended actions:

  1. Check the status of the NodeGroup:

     ```
     d8 k get ng NODE_GROUP_NAME -o yaml
     ```

     Look for errors in the `.status.lastMachineFailures` field.

  2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

```
d8 k get instances
```

3. If logs don’t show errors, and an Instance continues to be Pending, check its bootstrap status:

```
d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

5. If there’s no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeGroupReplicasUnavailable CE S8
  ▼ **Too many unavailable instances in the NODE_GROUP_NAME NodeGroup.**

  The number of simultaneously unavailable instances in the NODE_GROUP_NAME NodeGroup exceeds the allowed threshold. This may indicate that the autoscaler has provisioned too many nodes at once. Check the state of the Machine in the cluster. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

  1. Cloud provider resource limits.
  2. No access to the cloud provider API.
  3. Misconfiguration of the cloud provider or instance class.
  4. Problems with bootstrapping the Machine.

  Recommended actions:

  1. Check the status of the NodeGroup:

```
d8 k get ng NODE_GROUP_NAME -o yaml
```

     Look for errors in the `.status.lastMachineFailures` field.

  2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

```
d8 k get instances
```

  3. If logs don’t show errors, and an Instance continues to be Pending, check its bootstrap status:

```
d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
```

  4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

  5. If there’s no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeGroupReplicasUnavailable CE S8
  ▼ **NodeGroup NODE_GROUP_NAME has unavailable instances.**

  There are VALUE unavailable instances in the NodeGroup NODE_GROUP_NAME. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

1. Cloud provider resource limits.
2. No access to the cloud provider API.
3. Misconfiguration of the cloud provider or instance class.
4. Problems with bootstrapping the Machine.

Recommended actions:

1. Check the status of the NodeGroup:

   ```
   d8 k get ng NODE_GROUP_NAME -o yaml
   ```

   Look for errors in the `.status.lastMachineFailures` field.

2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

   ```
   d8 k get instances
   ```

3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

   ```
   d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
   ```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

   ```
   bootstrapStatus:
     description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
     logsEndpoint: http://192.168.199.158:8000
   ```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeGroupUsesDeprecatedMCMEngine CE S9
  ▼ NodeGroup NAME is managed by the deprecated Machine Controller Manager (MCM).

  The nodes of NodeGroup `NAME` are managed by Machine Controller Manager (MCM): `status.engine` is `MCM`. MCM is being replaced by Cluster API (CAPI) for the cloud provider of this cluster.

  To resolve this issue, move the group to CAPI by recreating it completely:

  1. Create a new NodeGroup with the same settings under a different name.
  2. Wait until the nodes of the new group are `Ready` and drain the nodes of NodeGroup `NAME`.
  3. Delete NodeGroup `NAME`.

  Check the engine of the groups:

  ```
  d8 k get nodegroups -o custom-columns=NAME:.metadata.name,ENGINE:.status.engine
  ```

- NodeRequiresDisruptionApprovalForUpdate CE S8
  ▼ Node NODE_NAME requires disruption approval to proceed with the update.

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, received initial approval, and started the update. However, it encountered a stage that may cause downtime and requires manual disruption approval, because the NodeGroup is configured with `disruptions.approvalMode: Manual`.

  To resolve the issue, ensure the Node is ready for unsafe updates (drained) and grant disruption approval by annotating the Node with `update.node.deckhouse.io/disruption-approved=`.

  Caution:

  - Nodes in manual mode aren't drained automatically.
  - Do not drain master nodes.

  1. To drain the Node and grant the update approval, run the following command:

     ```
     d8 k drain NODE_NAME --delete-emptydir-data --ignore-daemonsets --force=true &&
       d8 k annotate node NODE_NAME update.node.deckhouse.io/disruption-approved=
     ```

2. Uncordon the Node once the update is complete and the annotation `update.node.deckhouse.io/approved` is removed:

```
while d8 k get node NODE_NAME -o json | jq -e '.metadata.annotations |
has("update.node.deckhouse.io/approved")' > /dev/null; do sleep 1; done
d8 k uncordon NODE_NAME
```

If the NodeGroup has multiple Nodes, repeat this process for each one, since only one Node is updated at a time. Consider temporarily switching to automatic disruption approval (`disruptions.approvalMode: Automatic`).

- NodeStuckInDraining CE S6
  ▼ Node NODE_NAME keeps failing to drain.

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` is stuck in the draining process.

  The Node has the error from the last attempt in the `update.node.deckhouse.io/drain-failed` annotation, truncated to 1 KB. To read it, run:

```
d8 k get node NODE_NAME -o json | jq -r
'.metadata.annotations["update.node.deckhouse.io/drain-failed"]'
```

  The full error text is in the `DrainFailed` events on the Node:

```
d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=DrainFailed --
sort-by='.metadata.creationTimestamp'
```

  The drain is retried until it succeeds, so the alert clears on its own once the cause is gone. Until then the Node holds the update slot of its NodeGroup, and no other Node in that group is updated.

  The usual causes are a PodDisruptionBudget that never allows an eviction and a pod that never terminates. Fix the cause named in the error above.

- NodeStuckInDrainingForDisruptionDuringUpdate CE S6
  ▼ Node NODE_NAME is stuck in draining.

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, requested and received approval, started the update, and reached a step that could cause downtime. It is currently stuck in the draining process while waiting for automatic disruption approval.

  To get more details, run the following:

```
d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=ScaleDown --
sort-by='.metadata.creationTimestamp'
```

- StandbyHolderDeploymentReplicasUnavailable CE S5
  ▼ No available replicas in Deployment standby-holder-NODE_GROUP_NAME.

  Deckhouse has detected that there are no available replicas in the `standby-holder-NODE_GROUP_NAME` Deployment in the `d8-cloud-instance-manager` namespace.

  Check the Deployment and associated Pods for issues.

- UnmetCloudConditions CE S6
  ▼ Deckhouse update is unavailable due to unmet cloud provider conditions.

  Deckhouse has detected that some cloud provider–specific conditions have not been met. Until these issues are resolved, updating to the new Deckhouse release is not possible.

  Troubleshooting details:

  - Name: NAME
  - Message: MESSAGE_CONTENTS

Module okmeter

- D8OkmeterAgentPodIsNotReady CE S6

  Okmeter agent is not Ready.

Module openvpn

- OpenVPNClientCertificateExpired CE S4
  ▼ **OpenVPN client certificate expired for XXX**

  The OpenVPN client certificate for **XXX** has expired.

  Check the certificate information and make sure it actually expires:

  ```
  d8 k -n d8-openvpn get secrets -l name="XXX"  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

  Renew or delete the expired certificate. Check the documentation for details.

- OpenVPNClientCertificateExpired CE S5
  ▼ **OpenVPN client certificate expires for XXX in less than 7 days.**

  The OpenVPN client certificate for **XXX** is set to expire in VALUE days.

  Renewal of certificate required. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=XXX  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

- OpenVPNClientCertificateExpired CE S6
  ▼ **OpenVPN client certificate expires for XXX in less than 30 days.**

  The OpenVPN client certificate for **XXX** is set to expire in VALUE days.

  Renewal of certificate required. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=XXX  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCACertificateExpired CE S4
  ▼ **OpenVPN CA certificate has expired.**

  OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets openvpn-pki-ca  -o jsonpath='{.data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

  The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiresTomorrow CE S4
  ▼ **OpenVPN CA certificate will expire tomorrow**

  OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets openvpn-pki-ca  -o jsonpath='{.data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

  The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiringInAWeek CE S5
  ▼ **OpenVPN CA certificate expires in less than 7 days.**

  The OpenVPN CA certificate is set to expire in VALUE days.

  Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

  View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

- OpenVPNServerCACertificateExpiringSoon CE S5
  ▼ **OpenVPN CA certificate expires in less than 30 days.**

  The OpenVPN CA certificate is set to expire in VALUE days.

  Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCertificateExpired CE S4
  ▼ **OpenVPN server certificate has expired.**

  The OpenVPN server certificate has expired.

  OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  Check the certificate information and make sure it actually expires.

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCertificateExpiresTomorrow CE S4
  ▼ **OpenVPN server certificate will expire tomorrow.**

  The OpenVPN server certificate is set to expire in less than 1 day.

  OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  Check the certificate information and make sure it actually expires:

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

Module registry

- D8RegistryAgentNotReady CE S4
  ▼ **The registry agent on node NODE_NAME is not ready.**

  The registry agent on node `NODE_NAME` is not answering, or is not running at all. Every pull on the node goes through it, and without it containerd pulls straight from the registry an image names. Images of the platform itself, and of registries that need the cluster’s credentials or certificate authority, then fail on that node; other images still pull, bypassing the in-cluster cache and any `RegistryUpstream`.

  The kubelet restarts an agent that stops answering for a minute, so a state that lasts this long is one a restart does not fix: an agent that fails on start, or one that is not on the node at all.

  What the pod says:

  ```
  d8 k -n kube-system get pod registry-agent-NODE_NAME -o wide
  d8 k -n kube-system describe pod registry-agent-NODE_NAME
  d8 k -n kube-system logs registry-agent-NODE_NAME --previous --tail=50
  ```

  An agent that exits on start names in its log what it could not use, usually its certificate material under `/etc/kubernetes/registry-agent/pki` or containerd’s drop-in directory.

  No such pod means either that the kubelet has no manifest for it — on a node bashible configures, that is `/etc/kubernetes/manifests/registry-agent.yaml` — or that the API server refused the mirror pod, in which case the agent may well be running. The kubelet’s log on the node says which:

```
journalctl -u kubelet --since "1 hour ago" | grep registry-agent
```

A node of a NodeGroup with `systemType: DeckhouseEngine` has no SSH: start a debug pod on it with `d8 k debug node/NODE_NAME -it --image=ubuntu` and run `chroot /host` first.

Fifteen minutes, so that a node taking over the agent after the module starts managing it, or an update that replaces the agent on every node, does not raise it.

- D8RegistryAgentRestartingTooOften CE S5
  ▼ **The registry agent on node NODE_NAME keeps restarting.**

  The registry agent on node `NODE_NAME` has restarted five times or more in the last hour. Each restart drops every pull in flight on the node, and until the agent is back containerd pulls without it, where images of the platform cannot be pulled.

  Either the agent exits on its own, or the kubelet kills it because it stopped answering its liveness probe for a minute. The last state of the container says which:

  ```
  d8 k -n kube-system get pod registry-agent-NODE_NAME -o
  jsonpath='{.status.containerStatuses[0].lastState}' | jq
  d8 k -n kube-system logs registry-agent-NODE_NAME --previous --tail=100
  d8 k -n kube-system get events --field-selector involvedObject.name=registry-agent-NODE_NAME
  ```

  `Liveness probe failed` in the events is an agent that hung: the log before the restart shows what it was doing. An exit code with an error in the log is an agent that gave up: the log names what it could not use.

- D8RegistryAirGapTransitionHeld CE S5
  ▼ **The registry upstream cannot be removed yet.**

  The configuration has no upstream, so the cluster is meant to be air-gapped, and the module is still holding the previous upstream: either the cache is not complete enough to stand alone, or the cache leader’s report was not written by the `syncer` running on its node now, or workloads in the platform namespaces (`kube-system` and `d8-*`) still reference the upstream and would have nowhere to pull from without it. Workloads in other namespaces are not counted and do not hold the transition.

  This is the safe outcome and the cluster is pulling images normally. It is also the one transition in this module that could cut every node off from images, which is why it waits rather than proceeds — and why a wait this long is worth looking at: nothing about it resolves on its own if the cache has stopped filling, or if a rollout onto the in-cluster address is stuck.

  Check the cache and the current count of references to the upstream (`upstreamReferences`):

  ```
  d8 k get registrystorage registry -o jsonpath='{.status}' | jq
  '{phase,fill,safeToDropUpstream,upstreamReferences,leader}'
  d8 k get registryconfig registry -o jsonpath='{.status.effectiveUpstream}' | jq
  ```

  The `registry-controller` log names up to five of the referencing objects each time the count changes. To see the last such line:

  ```
  d8 k -n d8-system logs -l app=registry-controller -c registry-controller --prefix --tail=-1 | grep
  'still names the upstream' | tail -1
  ```

  If the leader is full and `upstreamReferences` is `0`, the upstream is held on the leader’s report: its `syncerStartedAt` is missing or earlier than the start of the `syncer` container in the leader’s storage pod. The `syncer` stamps its report once it has read its store, normally within two minutes of starting. The hold does not end while the leader’s storage pod is not Ready, or while that pod runs a `syncer` from before `syncerStartedAt` existed (during a module update), which stamps nothing. To see the leader’s report and the controller’s reason:

  ```
  d8 k get registrystorage registry -o json | jq '.status as $s | $s.replicas[] | select(.node ==
  $s.leader) | {node,full,error,syncerStartedAt}'
  d8 k -n d8-system logs -l app=registry-controller -c registry-controller --prefix --tail=-1 | grep
  'the upstream is held' | tail -1
  ```

  Deleting the leader’s storage pod is safe while the upstream is held: the replacement `syncer` reads its store and stamps its report, normally within two minutes.

- D8RegistryConfigInvalid CE S4
  ▼ **The registry configuration was rejected.**

The `registry` module cannot use its configuration, so the cluster keeps whatever arrangement it already had. Nothing has broken yet; nothing new will take effect either.

What was wrong is in the resource:

```
d8 k get registryconfig registry -o jsonpath='{.status.conditions}' | jq
```

- D8RegistryControllerDown CE S5
  ▼ The registry controller is down or cannot be scraped.

  No replica of `registry-controller` in `d8-system` is both ready and answering Prometheus. Either every replica is down or not ready, or they run and their metrics cannot be read — through `kube-rbac-proxy`, which needs a TokenReview to let Prometheus in.

  Pulls are not affected: every node keeps pulling the way its agent was last told. What stops is everything that changes that. A change to the `registry` ModuleConfig, a new RegistryUpstream, a leader moving to another master, an air-gap transition — none of it reaches the nodes until the controller is back.

  Beyond the registry, no namespace in the cluster can finish deleting: the controller also serves the `registry.deckhouse.io` API (the `RegistryImage` listing), and while nobody answers for it, discovery of that group fails and namespaces stay `Terminating` with the condition `NamespaceDeletionDiscoveryFailure`. They are deleted once a replica answers again.

  And while this is firing, most other `D8Registry*` alerts are blind: they read what the controller exports, so they stay silent whatever happens in the meantime. The exceptions are `D8RegistryStorageLeaderAbsent`, which reads the cache replicas themselves, and the migration and drain alerts, which Deckhouse feeds; those keep working.

  ```
  d8 k -n d8-system get deploy registry-controller
  d8 k -n d8-system get pods -l app=registry-controller -o wide
  d8 k -n d8-system logs deploy/registry-controller -c registry-controller --all-pods=true --tail=100
  d8 k -n d8-monitoring get podmonitor registry-controller
  ```

  If the pods are ready, the problem is in the scrape: look at the `kube-rbac-proxy` container's log for authentication errors.

- D8RegistryDrainStuck CE S6
  ▼ The registry module cannot finish leaving the pull path.

  The `registry` ModuleConfig asks for `Unmanaged`, and the module is still running its components and still serving `registry.d8-system.svc:5001`. That is deliberate: workloads in the cluster still name that address, and taking it away before they move would leave them unable to pull. It has been waiting for half an hour, which is longer than it should take.

  Nothing is degraded while this alert is firing. The cluster pulls, the module serves, and the only thing not happening is the withdrawal.

  What to look at:

  1. What still names the address. The module logs the first few on every pass, and the count is a metric:

     ```
     d8 k -n d8-system logs deploy/deckhouse -c deckhouse | grep "still serving the in-cluster
     registry"
     ```

  2. Then the workloads themselves, which is where the answer usually is:

     ```
     d8 k get pods,deployments,daemonsets,statefulsets -A -o json \
       | jq -r '.items[] | select([.spec.template.spec.containers[]?.image,
     .spec.containers[]?.image] | any(startswith("registry.d8-system.svc:5001"))) | "\(.kind) \
     (.metadata.namespace)/\(.metadata.name)"'
     ```

  A third cause is a cluster with nowhere to go, and it is the one to check first on a cluster that was air-gapped or installed with no upstream. Leaving the pull path means pointing the cluster back at the registry the module was fetching from, and a module serving purely from its own store has no such registry to name. It says so in the log:

  ```
  d8 k -n d8-system logs deploy/deckhouse -c deckhouse | grep "no upstream to point the cluster back
  at"
  ```

  Give the cluster a registry to pull from, or ask for `Managed` again. Nothing is degraded while it

waits, and the withdrawal will not complete until one of the two happens.

Otherwise two causes account for it. Either a release is not being re-rendered — check that the operator queue is not stuck, since the references move only as each module's release is rendered again:

```
d8 k -n d8-system exec deploy/deckhouse -c deckhouse -- deckhouse-controller queue main
```

Or something outside the platform's own renders names the address: a workload of yours that hardcoded it, or a manually edited manifest. Those will never move on their own, and until they are changed to pull from the registry the cluster was installed with, the module will keep serving them — which is the intended behaviour, not a failure.

One shape of this is worth naming, because it was hit the first time this ran on a live cluster: a rolling update stuck for reasons of its own. The new pod is already on the upstream registry and never becomes ready, so the superseded ReplicaSet keeps its old pod, and that pod names the in-cluster address. A pod that has already pulled is not waited on, so this alone does not hold the withdrawal — but a pod stuck BEFORE its containers start does, and rightly, since it is exactly what would fail. Fix the rollout, or delete the pod that cannot start:

```
d8 k get pods -A --field-selector status.phase=Pending
```

A drain can also be abandoned by asking for `Managed` again: the module then keeps the pull path it never finished giving up, and this alert goes away.

- D8RegistryMigrationPending CE S8
  ▼ **The registry module is held on its v1 implementation.**

  This cluster still pulls images the way the v1 implementation of the `registry` module arranged it, and the handover to the current implementation has not happened. This alert does not resolve on its own while the reason stays, and the following Deckhouse Platform releases stay blocked by the `registryImplementation` requirement.

  Reason: `XXX`

  The handover waits because both implementations configure the same thing on every node: which registry the container runtime asks, and with which credentials. Running both would not merge those two answers, it would race them. So the current implementation takes over only from a state it can take over from.

  What to do depends on the reason:

  - The reason names the **`Direct`** mode. This release can take over on its own once it knows where images come from: set `mode: Managed` in the `registry` ModuleConfig together with `primary.upstream`. Take the values from the `registry.direct` section of the `deckhouse` ModuleConfig, as the module's FAQ shows for the `Direct` mode: `imagesRepo` splits into `host` and `path`, the credentials stay the same, plus `ca` if the registry requires one. The handover happens on the module's next reconciliation and this alert clears.
  - The reason names the **`Proxy`** or **`Local`** mode, says the v1 implementation is **transitioning**, has **not recorded a mode**, or that its state or node configuration **cannot be read**. This state is not supported. A cluster gets here only through an update that bypassed the `registryImplementation` release requirement: a forced release, an edited image, or a jump from a release older than Deckhouse Platform 1.77.1. Deckhouse Platform 1.78 has no code that can manage or migrate this state.

    Deckhouse Platform 1.78 does not deploy the v1 implementation, and the update removed the components that managed its node services on the masters. Depending on the timing of the update, those services were stopped, or keep running with nothing to manage them or renew their certificates. The nodes keep the registry configuration the v1 implementation gave them. Images already on a node keep working; pulling any other image may fail.

    Do not change the `registry` or `deckhouse` ModuleConfigs and do not delete the `registry-*` secrets in `d8-system`: those secrets keep the handover closed, so that the current implementation is not deployed next to what the v1 implementation left. Contact Deckhouse Platform technical support.

  To see the mode the v1 implementation recorded:

  ```
  d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
  ```

  If that secret is gone, the node configuration the v1 implementation left still shows the mode. This prints the mode and the proxy endpoints only, without credentials:

```
d8 k -n d8-system get secret registry-bashible-config -o jsonpath='{.data.config}' | base64 -d \
  | awk '/^(agent|mode):/ {print; next} /^proxyEndpoints:/ {p=1; print; next} p && /^ *- / {print;
next} {p=0}'
```

- D8RegistryMirrorRejected CE S6
  ▼ **A mirror of the image set was refused.**

  A `RegistryMirror` resource was not accepted, so the nodes it selects pull the way they would without it: from the in-cluster cache or the upstream, however far those are.

  Reason: `XXX`

  - `NodeConflict` — a node it selects is served by a mirror created earlier. A node is served by one mirror at most; narrow the `nodeSelector` of one of them.
  - `InvalidSpec` — a field of `spec` is invalid.

```
d8 k get registrymirrors -o wide
```

- D8RegistryNodeForeignRegistryConfig CE S5
  ▼ **Node NODE_NAME carries container runtime registry configuration the registry module did not write.**

  Node `NODE_NAME` has a file under `/etc/containerd/conf.d/` (containerd v1) or `/etc/containerd/conf2.d/` (containerd v2) that configures a registry, and the `registry` module manages this cluster's registry configuration. Two writers of that configuration is not a state the platform supports, so bashible refuses to merge such a file.

  One exception for now: on containerd v1 with the module in the `Unmanaged` mode, a `conf.d` file is still merged. The refusal described below starts there once the module is switched to `mode: Managed`, so fix the file before that switch.

  Right now nothing is broken, and that is the danger. The refusal happens inside the bashible step that builds the container runtime configuration, and that step only runs when something makes it run. Until then the file has no effect at all: bashible compares the node's configuration checksum, finds it unchanged and reports "Configuration is in sync, nothing to do".

  The step does run on its own, though: a node re-applies its whole configuration periodically (roughly every four hours), and also on any registry change, platform upgrade or reboot. At that point the step fails with

```
Failed to merge /etc/containerd/conf.d/file: it contains custom registry fields,
configure them in /etc/containerd/registry.d instead
```

  and retries, so the node stops applying configuration entirely — hours after the change that caused it, and with nothing else to connect the two.

  What to do, before that happens:

  1. Find the file on the node:

```
ls /etc/containerd/conf.d/ /etc/containerd/conf2.d/ 2>/dev/null
grep -l registry /etc/containerd/conf.d/*.toml /etc/containerd/conf2.d/*.toml 2>/dev/null
```

  2. Move the registry part of it to a host directory, which is where this platform configures registries and where the module's node agent leaves other people's entries alone:

```
/etc/containerd/registry.d/registry host/hosts.toml
```

  3. Remove the registry section from the `conf.d` file (anything non-registry in it can stay), and remove the `NodeGroupConfiguration` that writes it if that is where it comes from.

  Doing it across a fleet with a `NodeGroupConfiguration` has one catch worth knowing in advance: give it a **weight below 32**. The step that refuses the file is `032_configure_containerd`, and a node stops at the step that fails — so a repair placed at a later weight never runs on the nodes that need it most. Measured: a cleanup configuration at weight 10 reached both nodes and cleared the label in 90 seconds.

  A node already stuck in the retry loop recovers on its own once the file is gone: measured, the configuration checksum returned in 75 seconds.

- D8RegistryNodeNotConverged CE S5
  ▼ **Some nodes have not applied their registry configuration.**

The agent on one or more nodes has not applied the layout it was given. Those nodes keep pulling images the way they were configured before, so this is not an outage — but a change made to the registry configuration has not reached them, and the next one will not either.

Which nodes, and what their agents say:

```
d8 k get registrynodes -o custom-columns=\
NODE:.metadata.name,GEN:.metadata.generation,APPLIED:.status.observedGeneration,OK:.status.reconciled

d8 k -n kube-system logs -l component=registry-agent --tail=50
```

Twenty minutes, because the agent re-reads its layout on a timer and a node that is merely slow or briefly unreachable resolves this on its own.

- D8RegistryNodeRunningFromDisk CE S6
  ▼ **Some nodes are routing pulls from their on-disk copy of the configuration.**

  The agent on one or more nodes read its layout from the API server and could not use it, so it is routing pulls from the copy it keeps on disk. Most often the credentials the layout references cannot be read; a read that failed while the status write after it went through looks the same. That is the copy working as designed — those nodes pull images normally, which is exactly why this is worth saying out loud: from every other angle it looks like success, and their configuration can drift arbitrarily far behind the cluster's.

  An hour, because a node that falls back for a pass or two is not news; a node that has not come off the copy since is.

  ```
  d8 k get registrynodes -o json | jq -r '
    .items[] | select(.status.conditions[]? | select(.type=="Reconciled" and
  .reason=="AppliedFromCache"))
    | .metadata.name'
  d8 k -n kube-system logs -l component=registry-agent --tail=200 | grep 'cannot resolve the
  credentials'
  ```

  What this alert cannot see is an agent that cannot write its status at all, since that write is how it reports the fallback. A node cut off from the API server is the main case: the agent reaches the API server the way the kubelet does, so the node shows as `NotReady` and is reported as such. An agent whose own access is refused is the other, and while its node stays `Ready` nothing reports it.

- D8RegistryStaleCacheData CE S8
  ▼ **A node still holds registry cache data that nothing uses.**

  Node `NODE_NAME` holds data of the in-cluster registry cache while no cache is configured. Nothing reads it, and nothing will unless the cache is turned back on.

  It is kept on purpose. Turning the cache back on refills from what is already there rather than from scratch, and over a slow link that difference is hours — so the module does not delete it, because that would make the decision irreversible in the one direction that hurts. What it will not do is keep it quietly, which is what this alert is.

  Nothing is wrong and nothing needs doing urgently. To reclaim the space, remove the contents of the directory the store kept its blobs in: `/var/lib/deckhouse/registry` on a node of a NodeGroup with `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` on other nodes. Keep the directory itself: the agent has it mounted, and once it is removed the agent no longer sees the node's directory, so data a cache leaves there later goes unreported until the agent restarts. With the cache off there is no RegistryStorage left to ask, so the command below looks at both.

  Open a root shell on the node over SSH. A node of a NodeGroup with `systemType: DeckhouseEngine` has no SSH: start a debug pod on it and switch to the node's root filesystem:

  ```
  d8 k debug node/NODE_NAME -it --image=ubuntu
  chroot /host
  ```

  Then remove the data, keeping the directories themselves:

  ```
  for dir in /opt/deckhouse/registry /var/lib/deckhouse/registry; do
    [ -d "$dir" ] && du -sh "$dir" && find "$dir" -mindepth 1 -delete
  done
  ```

  Twenty-four hours, so that turning the cache off and on again in one working session never raises it.

- D8RegistryStorageIncomplete CE S6
  ▼ **Some registry cache replicas do not hold the whole image set.**

  One or more replicas of the in-cluster cache are still filling, or have stopped short. While an upstream is configured this costs nothing at pull time — a node falls back to it — but it does mean the cluster could not survive losing that upstream, and it is what holds an air-gap transition back.

  Two hours, because filling a cache over a slow link legitimately takes that long.

  A replica that reports an `error` counts as not holding the set, whatever it holds: the error says why — a store that is not on the disk meant for it, for one. A replica whose store the registry cannot write to is left out: it is neither filling nor going to, and `D8RegistryStoreWritesRefused` reports it with what to do.

  ```
  d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, full, error}'
  ```

- D8RegistryStorageLeaderAbsent CE S4
  ▼ **The registry cache has no leader.**

  No ready replica of the in-cluster registry cache (`registry-storage` in `d8-system`) reports holding the storage lease.

  The leader is where the nodes pull from, and the only replica that writes. Without it nothing fills the cache, the followers have nobody to replicate from, and `d8 mirror push` has nowhere to go. The agents still route pulls to the replica named as leader in their layout and wait for it before anything else: with an upstream configured a pull is slower and comes from the upstream; in an air-gapped cluster an image a node does not already hold cannot be pulled at all.

  Which of three things it is decides what to do, so look first:

  ```
  d8 k -n d8-system get pods -l app=registry-storage -o wide
  d8 k -n d8-system get lease registry-storage -o jsonpath='{.spec.holderIdentity} {.spec.renewTime}
  {"\n"}'
  d8 k get registrystorage registry -o jsonpath='{.status.leader}{"\n"}'
  d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, role, full, error}'
  ```

  - The Lease is held and its `renewTime` keeps moving: the leader is alive, but its report is not counted. Look at the replica on the node named in `holderIdentity`:

    ```
    d8 k -n d8-system get pods -l app=registry-storage -o wide --field-selector
    spec.nodeName=holderIdentity
    d8 k -n d8-system describe pod the pod from the line above
    ```

    If it is not ready, Prometheus does not scrape it, since it scrapes only ready pods. `syncer` has no readiness probe; it is usually the `distribution` or `auth` container that is not ready, and a leader with either of them not ready serves pulls badly or not at all. Find out which one fails and why. If the pod is ready, its scrape fails while the others' do not: look at its `kube-rbac-proxy` container's log for errors.

  - The Lease's `renewTime` has stopped while other replicas are ready: nobody leads. First rule out a master that has gone down: the Lease's `holderIdentity` above is the node the last leader ran on.

    ```
    d8 k get nodes -l node-role.kubernetes.io/control-plane
    ```

    If a master is down and is not coming back, delete its Node object: the replica that ran on it then no longer counts in the election. Otherwise, or if the alert keeps firing after that, look at the `error` each replica reports in the list above and at the `syncer` logs of the replicas that are up:

    ```
    d8 k -n d8-system logs registry-storage-0 -c syncer --tail=100
    ```

  - No replica is ready, or none can be scraped. Then this is not about the election: the pods are down or not ready (where the `extended-monitoring` module is enabled, `KubernetesStatefulSetReplicasUnavailable` says so as well), or Prometheus cannot read them through `kube-rbac-proxy`, and the leader may well be alive. The Lease above tells it: a `renewTime` that keeps moving means somebody holds it.

- D8RegistryStorageNotReclaimed CE S6
  ▼ **The registry cache has not reclaimed its disk for a week.**

  No replica of the in-cluster cache has finished a garbage collection in the last week. Nothing is

broken yet: the cache serves images normally, and it will keep doing so until the disk fills.

It is worth attention because the collection is the only thing that ever removes anything. Every release adds a slice of the repository, so a store nobody reclaims grows in one direction — and a full store cannot be written to, which in an air-gapped cluster means it cannot be updated.

Where to look:

```
d8 k get registryoperation -o wide
d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq \
  'map({node, collectedAt, collectionError})'
d8 k get registrystorage registry -o jsonpath='{.spec.garbageCollection}' | jq
```

The likely reasons, in the order they are worth checking:

- the collection is turned off in the `registry` ModuleConfig under `storage.garbageCollection.enabled`;
- the schedule has not come round yet, or names an hour that never arrives — an unreadable expression is refused rather than guessed at — and the image set has not changed either;
- the collections are created and fail: the `RegistryOperation` objects of type `GarbageCollection` say why in `status.replicas`, for a day after they finish;
- no release is deployed, which is the one input that justifies deleting anything: every deletion is “older than what the cluster runs”, and without that there is nothing to compare against;
- `collectionError` on a replica, which says what its own last attempt hit.

One reading tells the first two apart from the rest. The controller logs every collection it creates and every firing of the schedule it skips, so their absence means none was ever started — a different problem from one that is started and failing:

```
d8 k -n d8-system logs -l app=registry-controller -c registry-controller --tail=-1 | grep -iE
"garbage collection|schedule fired"
```

Collecting is not an outage — the replica keeps serving every image it holds — but it pauses filling the replica’s store and can fail a `d8 mirror push` started meanwhile, so the schedule belongs at a quiet hour, which is where it defaults to.

- D8RegistryStorageUpdateBlocked CE S6
  ▾ **The in-cluster registry cache cannot be updated to its new revision.**

  The `registry-storage` StatefulSet has a new revision, and the controller that replaces its replicas has refused to replace the next one for an hour. Existing replicas remain running, but the cache update cannot proceed. The `reason` label identifies what is blocking it.

  Reason: `XXX`

  - `HolderNotReady` — the cache has one replica and no upstream, so its new images must be on its master before it is replaced. The `registry-storage-image-holder` DaemonSet keeps them there and has not got all of them yet. If its pod on the master is in `ImagePullBackOff`, the cache does not hold an image: push the Deckhouse Platform release the cluster is updating to with `d8 mirror push`, and the holder pulls it on its next retry. If the pod is in `CrashLoopBackOff` or `CreateContainerError`, the holder has the image but cannot run it, and a push does not help: the cause is in the events of the pod. Otherwise it is still rolling or pulling, and there is nothing to do.
  - `SingleReplicaWithoutHolder` — the same, and the DaemonSet has no pod on that master at all. For a moment this is normal while the DaemonSet creates its pod there; lasting an hour, it means the DaemonSet does not schedule a pod there. Find out why with `d8 k -n d8-system describe daemonset registry-storage-image-holder`.
  - `LeaderWithoutFullFollower` — only the leader is left to replace, and no follower holds the whole image set to take over from it. If the leader holds the set, the followers are copying it and the update continues once one of them finishes. If the leader reports that it does not hold it, the followers cannot complete the set either, since they copy it from the leader: pull the Deckhouse Platform release and the modules the cluster uses with `d8 mirror pull`, as described in the air-gapped update guide, and push them into the cache with `d8 mirror push`, as in step 2 of Going air-gapped. If the leader has not reported yet, reports an error or cannot tell which set the cluster needs, pushing does not help: the cause is in the log of the `syncer` container of the leader’s pod.
  - `LeaderNotServing` — the storage lease names a master where no serving replica runs. The replicas move the lease on their own once it expires.
  - `StorageUnknown` — the `RegistryStorage` resource does not exist. The controller creates it; check the log of `registry-controller`.

  The event on the StatefulSet says which replica and what it is waiting for:

```
d8 k -n d8-system get events --field-selector involvedObject.name=registry-
storage,reason=UpdateBlocked
```

And the two things it is waiting on:

```
d8 k -n d8-system get pods -l app=registry-storage-image-holder -o wide
d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq
```

- D8RegistryStoreCorrupted CE S5
  ▼ The registry cache holds broken blobs that were not repaired.

  A replica of the in-cluster cache holds blobs whose content does not match their digest, and they have not been repaired for two hours. Every pull of an image that uses such a blob from this replica fails; while an image of the cluster's set uses one, the replica is not full, and a leader hands the lease to a replica that holds the set whole.

  The module creates a `RegistryOperation` of type `Repair` for these blobs on its own, and repeats it four hours after one that did not fix them. Where to look:

  ```
  d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq 'map({node, full,
  corruptedBlobs})'
  d8 k get registryoperation -o wide
  d8 k get registryimages -l registry.deckhouse.io/state=Corrupted
  ```

  `status.replicas[].repair.digests[].message` of the last `Repair` says why a blob could not be repaired. The usual reasons:

  - no whole copy anywhere: the cache has one replica and no upstream, or every replica holds the blob broken — as after a `d8 mirror push` of a broken bundle. Push the release again with `d8 mirror push`, then create a `Repair` for the digests by hand;
  - another operation held the queue, such as a long garbage collection: the repair runs once it finishes;
  - the other replicas could not be reached from this one.

  The disk under the store is worth checking too: blobs that go bad on their own are a sign of a failing disk.

- D8RegistryStoreNearlyFull CE S6
  ▼ The registry cache on node NODE_NAME holds over 85% of storage.size.

  The in-cluster registry cache on node `NODE_NAME` holds more than 85% of its budget, `storage.size` in the `registry` ModuleConfig. Nothing is refused yet. The next release may not fit: once the budget is reached, the cache stops accepting writes and `D8RegistryStoreWritesRefused` follows.

  ```
  d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, store}'
  ```

  Raise `storage.size` if the disk under the store has room — `filesystemFreeBytes` above `reserveBytes` is what is left — or check that the garbage collection runs: it is the only thing that reclaims space, and `D8RegistryStorageNotReclaimed` says when it has not.

- D8RegistryStoreUnbounded CE S7
  ▼ The registry cache has no storage.size, so only the node's disk limits it.

  The in-cluster registry cache is on, and `storage.size` is not set in the `registry` ModuleConfig. The node is still protected: the store never takes the space kept free for the kubelet. But that is the only limit, so the store may grow until it leaves the node exactly that and no more — and on a master that shares its disk with the store, everything else on it then works with the minimum.

  Set `storage.size` to what the store may take on each master: enough for three releases of the image set together with the modules and the module catalog, and no more than fits on the disk under the store above the reserve:

  ```
  d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, store}'
  ```

  An hour, so that turning the cache on and setting the size in two steps does not raise it.

- D8RegistryStoreWritesRefused CE S4
  ▼ The registry cache on node NODE_NAME does not accept writes (XXX).

  The in-cluster registry cache on node `NODE_NAME` has stopped accepting writes, because `XXX`. It keeps serving every image it holds. What stops is everything that ADDS to it: filling the next

release, replicating from the leader, `d8 mirror push`. In an air-gapped cluster a release that is not in the cache cannot be installed.

What the replica measured, and the error it reports:

```
d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, store, error}'
```

`BudgetExhausted` — the store holds as much as `storage.size` allows. The node is safe: this is the limit doing its job, and images the store cannot keep are streamed from the upstream while there is one. Either raise `storage.size` in the `registry` ModuleConfig, if the disk has room for it, or let the garbage collection reclaim what no deployed release needs any more: it runs on `storage.garbageCollection.schedule` and keeps the deployed release, the previous one and everything newer.

`ReserveExhausted` — the filesystem under the store is down to the space kept free on it. The node is safe for the same reason, and images are streamed the same way. `storage.size` is not what stops it, the disk is. On a filesystem shared with the node (`dedicatedFilesystem` absent from the replica's `store`) that is the kubelet's eviction threshold plus a margin: free space on the node, or — which is what fixes it for good — give the store a disk of its own, sized for `storage.size`. On a disk of the store's own (`dedicatedFilesystem: true`) it is the margin alone, and the disk itself is full: grow it — the reserve follows without a restart — or let the garbage collection reclaim what no deployed release needs, as for `BudgetExhausted`.

`NotWritable` — the registry cannot create files in its tree in the store, whatever room the store has: `docker/registry/v2` and the directories under it, or the store's directory while that tree does not exist yet. It runs as root without capabilities, so those directories have to belong to root and let their owner write. Other causes are the immutable attribute, a read-only mount, or a security module such as SELinux or AppArmor. No limit is involved: neither the garbage collection nor a larger `storage.size` helps, and the collection does not start for it. Until it is fixed, a pull the cache would keep from the upstream fails with an error instead of being streamed. The replica's `error` names the directory on the node, its owner and mode, and the command that fixes it:

```
d8 k get registrystorage registry -o json | jq -r '.status.replicas[] | select(.store.reason ==
"NotWritable") | "\(.node): \(.error)"'
```

`NotWritable` takes precedence over a limit the store is also at. Once it is fixed, the replica says so within a minute, and the alert clears or names the limit.

- D8RegistryUpstreamProbeFailing CE S4
  ▼ **The configured registry upstream does not pass its preflight check.**

  A change to the primary upstream was refused, and the cluster is still using the last one that worked. Nothing is degraded — that is what holding the previous upstream is for — but the change has not taken effect and will not until the check passes.

  The `outcome` label says which of three things is wrong, and they are not the same problem:

  - `unreachable` — the registry did not answer at all: network, firewall, or the registry itself.
  - `auth` — it answered and rejected the credentials. Usually an expired or wrong license key.
  - `sentinel` — it answered and accepted the credentials, but does not hold the Deckhouse images. Usually the wrong repository path.

  Current outcome: `XXX`

  ```
  d8 k get registryconfig registry -o jsonpath='{.status.conditions}' | jq
  ```

- D8RegistryUpstreamRejected CE S6
  ▼ **An additional registry was refused.**

  A `RegistryUpstream` resource was not accepted, so pulls for the registry it names are not intercepted on any node. Whatever asked for it — a module, or an operator — is not getting it.

  Reason: `XXX`

  ```
  d8 k get registryupstreams -o custom-columns=\
  NAME:.metadata.name,MATCH:.spec.match,ACCEPTED:.status.conditions[0].status,REASON:.status.conditions[
  0].reason
  ```

Module secret-copier

- D8SecretCopierDeprecatedLabels CE S9
  ▼ **Obsolete antiopa_secret_copier=yes label detected.**

The `secret-copier` module has changed its labeling approach for original secrets in the `default` namespace.

The label `antiopa-secret-copier: "yes"` is deprecated and will be removed soon.

To resolve this issue, replace the label `antiopa-secret-copier: "yes"` with `secret-copier.deckhouse.io/enabled: ""` for all secrets used by the `secret-copier` module in the `default` namespace.

- D8SecretCopierSecretNotCopied CE S6
  ▼ **Secret XXX was skipped by secret-copier, reason XXX.**

  The `secret-copier` module skipped the Secret `XXX` with the reason `XXX`.

  The meaning of the `namespace` label, which is `NAMESPACE` here, depends on the reason:

  - `InvalidNamespaceSelector`: `namespace` is the namespace of the source Secret, `default`. The `secret-copier.deckhouse.io/target-namespace-selector` annotation of the source Secret is not a valid label selector, so the Secret is not copied to any namespace and its copies made before are deleted. Fix the annotation value. The exact parsing error is in the Deckhouse logs.
  - `ForeignSecretExists`: `namespace` is the target namespace. It already has a Secret `XXX` that was not created by `secret-copier`. Delete or rename it, and the copy will be created.

Module terraform-manager

- D8TerraformStateExporterClusterStateChanged CE S8
  ▼ **Terraform-state-exporter cluster state change detected.**

  The current Kubernetes cluster state is `STATUS_REFERENCE` compared to the Terraform state.

  It is important to reconcile the states.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterClusterStateError CE S8
  ▼ **Terraform-state-exporter cluster state error.**

  The `terraform-state-exporter` can’t check difference between the Kubernetes cluster state and the Terraform state.

  That was likely caused by `terraform-state-exporter` failing to run Terraform with the current state and configuration.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterHasErrors CE S8
  ▼ **Terraform-state-exporter has encountered errors.**

  Errors occurred during the operation of the `terraform-state-exporter`.

  To get more details, check the Pod logs:

  ```
  d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
  ```

- D8TerraformStateExporterNodeStateChanged CE S8
  ▼ **Terraform-state-exporter node state change detected.**

The current state of node `NODE_GROUP_NAME/NAME` is `STATUS_REFERENCE` compared to the Terraform state.

It is important to reconcile the states.

Troubleshooting steps:

1. View the differences:

   ```
   dhctl terraform check
   ```

2. Apply the necessary changes to bring the cluster in sync:

   ```
   dhctl converge
   ```

- D8TerraformStateExporterNodeStateError CE S8
  ▼ **Terraform-state-exporter node state error.**

  The `terraform-state-exporter` can’t check the difference between the node `NODE_GROUP_NAME/NAME` state and the Terraform state.

  Probably, it occurred because `terraform-manager` had failed to run Terraform with the current state and configuration.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterNodeTemplateChanged CE S8
  ▼ **Terraform-state-exporter node template change detected.**

  The `terraform-state-exporter` has detected a mismatch between the node template in the cluster provider configuration and the one specified in the NodeGroup NAME`.

  Node template is `STATUS_REFERENCE`.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Adjust NodeGroup settings to fix the issue or bring the cluster in sync via the following command:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterPodIsNotReady CE S8
  ▼ **Terraform-state-exporter Pod is not Ready.**

  The `terraform-state-exporter` cannot check the difference between the actual Kubernetes cluster state and the Terraform state.

  To resolve the issue, check the following:

  1. Deployment description:

     ```
     d8 k -n d8-system describe deployment terraform-state-exporter
     ```

  2. Pod status:

     ```
     d8 k -n d8-system describe pod -l app=terraform-state-exporter
     ```

- D8TerraformStateExporterPodIsNotRunning CE S8
  ▼ **Terraform-state-exporter Pod is not Running.**

The `terraform-state-exporter` cannot check the difference between the actual Kubernetes cluster state and the Terraform state.

To resolve the issue, check the following:

1. Deployment description:

   ```
   d8 k -n d8-system describe deployment terraform-state-exporter
   ```

2. Pod status:

   ```
   d8 k -n d8-system describe pod -l app=terraform-state-exporter
   ```

- D8TerraformStateExporterTargetAbsent CE S8
  ▼ **Terraform-state-exporter target is missing in Prometheus.**

  Prometheus cannot find the `terraform-state-exporter` target.

  To investigate the details:

  - Check the Pod status:

    ```
    d8 k -n d8-system get pod -l app=terraform-state-exporter
    ```

  - Check the container logs:

    ```
    d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
    ```

- D8TerraformStateExporterTargetDown CE S8
  ▼ **Prometheus can't scrape terraform-state-exporter.**

  Prometheus is unable to scrape metrics from the `terraform-state-exporter`.

  To investigate the details:

  - Check the Pod status:

    ```
    d8 k -n d8-system get pod -l app=terraform-state-exporter
    ```

  - Check the container logs:

    ```
    d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
    ```

Module user-authn

- D8DexAllTargetsDown CE S6
  ▼ **Prometheus is unable to scrape Dex metrics.**

  Prometheus cannot scrape any Dex pod in the `d8-user-authn` namespace. Users may be unable to sign in.

  Check the Dex pods and their logs:

  ```
  d8 k -n d8-user-authn get pods -l app=dex
  d8 k -n d8-user-authn logs deploy/dex
  ```

- D8DexAuthenticatorNotServed CE S6
  ▼ **The DexAuthenticator XXX/XXX is not served, the reason is XXX.**

  The DexAuthenticator `XXX` in the namespace `XXX` is not served, the reason is `XXX`. Users cannot sign in through it. An object that was served before works without its latest changes. An object that had a dex-authenticator of its own before the module update keeps it, and changes of `allowedGroups` and `allowedEmails` do not apply to it.

  Read the message of the condition:

```
d8 k -n XXX get dexauthenticator XXX -o jsonpath='{.status.conditions[?
(@.type=="Served")].message}'
```

What to do by the reason:

- `HostConflict`: The domain belongs to another DexAuthenticator. Change the domain, or the Ingress class if the domain is to be shared.
- `InvalidRecord`: The object cannot be served as it is. Fix it as the message says.
- `ClientNotCreated`: The module cannot register the application in Dex. The message gives the reason.
- `ForeignObject`: The namespace has a Service, an Ingress or an HTTPRoute with a name the module needs. Rename or delete it.
- `ObjectsNotWritten`: The API server refused the Service or a route of the object, for example by an admission policy. The message gives its answer.
- `NamespaceShareFull`: The condition has the reason `TableFull`: the namespace has as many DexAuthenticator objects as one namespace may have. Delete the ones that are no longer needed.

- D8DexAuthenticatorWaitingForInstance CE S6
  ▾ The DexAuthenticator XXX/XXX has been waiting for the shared dex-authenticator XXX for more than 30 minutes.

  For more than 30 minutes, the shared dex-authenticator instance `XXX` has not loaded the settings of the DexAuthenticator `XXX` in the namespace `XXX`. It normally takes a few minutes. Until then a new object is not served, and an object served before works without its latest changes.

  Check the pods of the instance and their log:

  ```
  d8 k -n d8-user-authn get pods -l app=shared-dex-authenticator,instance=XXX
  d8 k -n d8-user-authn logs -l app=shared-dex-authenticator,instance=XXX -c dex-authenticator --
  prefix --tail=-1
  ```

- D8DexAuthRequestsQuotaExhausted CE S3
  ▾ Dex AuthRequest objects ResourceQuota is exhausted in namespace "d8-user-authn".

  The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` has been fully exhausted.

  Dex is currently unable to create new AuthRequest custom resources, which will cause:

  - Dex health checks to fail immediately.
  - Authentication through external identity providers to stop working.
  - Users unable to log in via OAuth/OIDC flows.

  Immediate actions required:

  1. Check current AuthRequest objects and clean up stale ones:

     ```
     d8 k get authrequests.dex.coreos.com -n d8-user-authn --no-headers | wc -l
     d8 k delete authrequests.dex.coreos.com --all -n d8-user-authn
     ```

     Deleting AuthRequest objects interrupts sign-ins that are in progress.

  2. Temporarily increase the ResourceQuota limit:

     ```
     d8 k patch resourcequota authrequest-quota -n d8-user-authn --type='json' -p='[{"op":
     "replace", "path": "/spec/hard/count\/authrequests.dex.coreos.com",
     "value": "20000"}]'
     ```

     The module restores the quota on its next run.

  3. Investigate why Dex is not cleaning up AuthRequest objects automatically.

  4. AuthRequest objects expire after 10 minutes. A growing count usually means that a client or a bot repeatedly starts sign-ins. Find the source in the Dex logs.

- D8DexAuthRequestsQuotaHigh CE S4
  ▾ Dex AuthRequest objects quota usage is higher than 80% in namespace "d8-user-authn".

  The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` is approaching its limit.

Current usage: XXX%

When the quota is exhausted, Dex will be unable to create new AuthRequest custom resources, which may cause:

- Dex health checks to fail.
- Authentication through external identity providers (GitLab, GitHub, etc.) to stop working.

Investigation steps:

1. Check for stale AuthRequest objects that Dex should have cleaned up:

   ```
   d8 k get authrequests.dex.coreos.com -n d8-user-authn
   ```

2. AuthRequest objects expire after 10 minutes. A growing count usually means that a client or a bot repeatedly starts sign-ins. Find the source in the Dex logs.

3. Consider temporarily increasing the ResourceQuota limit if needed. The module restores the quota on its next run.

- D8DexConnectorNotInitialized CE S6
  ▾ **Dex connector "XXX" is not initialized.**

  At least one Dex pod failed to initialize connector `XXX` of type `XXX` during the last opening attempt for its current configuration. Sign-ins and token refreshes through this connector may fail on that pod.

  Check the Dex pods and their logs.

  ```
  d8 k -n d8-user-authn get pods -l app=dex
  d8 k -n d8-user-authn logs -l app=dex -c dex --prefix
  ```

  Check the provider address, credentials, TLS certificates, and network connectivity configured in the corresponding DexProvider. Dex retries initialization when the connector is used. A sign-in retries only on the pod handling the request. Other pods can retain a failed result until the connector is used there or they restart.

  After correcting the configuration or restoring the provider, restart Dex to retry initialization on every pod:

  ```
  d8 k -n d8-user-authn rollout restart deployment/dex
  d8 k -n d8-user-authn rollout status deployment/dex
  ```

  A rolling restart can interrupt in-flight sign-ins and token refreshes on the restarting pods.

  The alert stops when no pod reports a failed opening attempt for the current configuration, or the connector is removed. An updated configuration has unknown initialization status until its next opening attempt; it does not report a failure in the meantime.

- D8DexProviderLDAPTLSConflict CE S8
  ▾ **DexProvider NAME has a conflicting LDAP TLS configuration.**

  DexProvider `NAME` combines mutually exclusive LDAP TLS settings: `XXX`. Dex silently ignores the redundant flag, which masks misconfiguration:

  - `insecureNoSSL + startTLS`: TLS is disabled (ldap://, port 389), so `startTLS` is ignored.
  - `insecureNoSSL + insecureSkipVerify`: TLS is disabled, there is no certificate to verify.
  - `insecureNoSSL + rootCAData`: TLS is disabled, the CA chain is ignored.

  The CRD schema rejects this combination on create/update; this DexProvider was created before the validation was added and remains active.

  Update the DexProvider to remove the conflict:

  - Plaintext LDAP (port 389): keep `spec.ldap.insecureNoSSL: true`, remove `startTLS`, `insecureSkipVerify`, and `rootCAData`.
  - STARTTLS on port 389: set `spec.ldap.insecureNoSSL: false`, `spec.ldap.startTLS: true`. Use `insecureSkipVerify` or `rootCAData` as needed.
  - `ldaps://` on port 636: leave `insecureNoSSL` and `startTLS` at `false`. Use `insecureSkipVerify` or `rootCAData` as appropriate.

  Audit all LDAP DexProvider objects:

```
d8 k get dexproviders.deckhouse.io -o yaml | yq '.items[] | select(.spec.type=="LDAP") |
{name: .metadata.name, ldap: .spec.ldap |
pick(["insecureNoSSL","startTLS","insecureSkipVerify","rootCAData"])}'
```

- D8SharedDexAuthenticatorClientTableAlmostFull CE S6
  ▼ The table of clients of the shared dex-authenticator XXX is XXX full.

  The table of clients of the shared dex-authenticator instance `XXX` is XXX full. New DexAuthenticator objects are not served once it is full.

  Delete the DexAuthenticator objects that are no longer needed:

  ```
  d8 k get dexauthenticators -A
  ```

- D8SharedDexAuthenticatorClientTableFull CE S4
  ▼ The table of clients of the shared dex-authenticator XXX is full, VALUE DexAuthenticator objects did not fit into it.

  The table of clients of the shared dex-authenticator instance `XXX` is full. VALUE DexAuthenticator objects are not served, or are served without their latest changes.

  Delete the DexAuthenticator objects that are no longer needed. The objects that did not fit:

  ```
  d8 k get dexauthenticators -A -o json | jq -r '.items[] |
  select(any(.status.conditions[]?; .type == "Served" and .reason ==
  "TableFull")) | "\(.metadata.namespace)/\(.metadata.name)"'
  ```

- D8SharedDexAuthenticatorClientTableNotLoaded CE S6
  ▼ The pod POD_NAME of the shared dex-authenticator XXX cannot load its table of clients.

  For more than 10 minutes, the pod `POD_NAME` of the shared dex-authenticator instance `XXX` has failed to load its table of clients. The pod keeps serving the table it loaded before: a DexAuthenticator created or changed since then is not served by this pod, and a pod started without a table answers 403 for every application.

  Check that the table Secret exists and read the log of the pod:

  ```
  d8 k -n d8-user-authn get secret XXX-clients
  d8 k -n d8-user-authn logs POD_NAME -c dex-authenticator
  ```

- D8SharedDexAuthenticatorSessionStoreNotSeeded CE S9
  ▼ The pod POD_NAME of the shared dex-authenticator XXX started without the sessions of the other pod.

  The pod `POD_NAME` of the shared dex-authenticator instance `XXX` has not copied the sessions of the other pod since it started. Users stay signed in. If the other pod restarts before the sessions are refreshed, which takes up to `idTokenTTL`, every session is refreshed with an additional request to Dex.

  The alert is resolved an hour after the pod starts. Check the log of the pod:

  ```
  d8 k -n d8-user-authn logs POD_NAME -c dex-authenticator
  ```

- D8SharedDexAuthenticatorWithoutOwnRedis CE S6
  ▼ The pod POD_NAME of the shared dex-authenticator XXX does not use its own Redis.

  For more than 10 minutes, the pod `POD_NAME` of the shared dex-authenticator instance `XXX` has not used the Redis in its pod. Users stay signed in, but their sessions are refreshed with additional requests to Dex.

  Check the containers of the pod and its log:

  ```
  d8 k -n d8-user-authn get pod POD_NAME
  d8 k -n d8-user-authn logs POD_NAME -c dex-authenticator
  ```

- D8UserAuthnBasicAuthProxyCertExpired CE S5
  ▼ The basic-auth-proxy client certificate has expired.

  The client certificate used by basic-auth-proxy has expired. The API server rejects Kubernetes API

requests forwarded by the proxy with basic authentication (`enableBasicAuth` in a DexProvider). Deckhouse has failed to renew this certificate.

After a failed attempt, the running Deckhouse pod waits at least 50 minutes before retrying while the current certificate remains valid, or at least 5 minutes if it is missing or expired. The hook checks every 5 minutes and on module runs. The certificate is signed by a `proxy-cert-generate-job-*` Job in the `d8-system` namespace on a control plane node. Deckhouse keeps a failed Job for inspection and removes it during a later attempt or through the Job TTL. Inspect the Job status, warning events, and Deckhouse log to identify the cause.

```
d8 k -n d8-system get jobs -l name=proxy-cert-generate-job
d8 k -n d8-system get events --field-selector type=Warning | grep proxy-cert-generate-job
d8 k -n d8-system logs deploy/deckhouse | grep 'basic-auth-proxy client certificate'
```

The alert stops after a valid replacement certificate is issued.

- D8UserAuthnBasicAuthProxyCertMissing CE S5
  ▼ **The basic-auth-proxy client certificate is missing.**

  Deckhouse has failed to issue a client certificate for basic-auth-proxy. basic-auth-proxy is not deployed, and the published Kubernetes API cannot accept basic authentication (`enableBasicAuth` in a DexProvider) until a certificate is issued.

  After a failed attempt, the running Deckhouse pod waits at least 50 minutes before retrying while the current certificate remains valid, or at least 5 minutes if it is missing or expired. The hook checks every 5 minutes and on module runs. The certificate is signed by a `proxy-cert-generate-job-*` Job in the `d8-system` namespace on a control plane node. Deckhouse keeps a failed Job for inspection and removes it during a later attempt or through the Job TTL. Inspect the Job status, warning events, and Deckhouse log to identify the cause.

  ```
  d8 k -n d8-system get jobs -l name=proxy-cert-generate-job
  d8 k -n d8-system get events --field-selector type=Warning | grep proxy-cert-generate-job
  d8 k -n d8-system logs deploy/deckhouse | grep 'basic-auth-proxy client certificate'
  ```

  The alert stops after a client certificate is issued.

- D8UserAuthnBasicAuthProxyCertRenewalFailed CE S5
  ▼ **Deckhouse cannot renew the basic-auth-proxy client certificate.**

  Deckhouse has failed to renew the client certificate used by basic-auth-proxy to forward Kubernetes API requests with basic authentication (`enableBasicAuth` in a DexProvider). The proxy continues using its current certificate. When this certificate expires, the API server rejects forwarded requests.

  After a failed attempt, the running Deckhouse pod waits at least 50 minutes before retrying while the current certificate remains valid, or at least 5 minutes if it is missing or expired. The hook checks every 5 minutes and on module runs. The certificate is signed by a `proxy-cert-generate-job-*` Job in the `d8-system` namespace on a control plane node. Deckhouse keeps a failed Job for inspection and removes it during a later attempt or through the Job TTL. Inspect the Job status, warning events, and Deckhouse log to identify the cause.

  ```
  d8 k -n d8-system get jobs -l name=proxy-cert-generate-job
  d8 k -n d8-system get events --field-selector type=Warning | grep proxy-cert-generate-job
  d8 k -n d8-system logs deploy/deckhouse | grep 'basic-auth-proxy client certificate'
  ```

  The alert stops after a renewed certificate is issued.

- D8UserAuthnControllerReconcileErrorsHigh CE S8
  ▼ **JOB_NAME reconcile errors are high for CONTROLLER_NAME.**

  Controller `CONTROLLER_NAME` of `JOB_NAME` has a reconcile error rate above 0.1 errors per second for more than 10 minutes.

  Check logs:

  ```
  d8 k -n d8-user-authn logs deploy/JOB_NAME --all-pods=true
  ```

- D8UserAuthnControllerTargetDown CE S6
  ▼ **Prometheus is unable to scrape user-authn-controller.**

Prometheus cannot scrape metrics from `user-authn-controller` in namespace `d8-user-authn`.

The controller is down, not Ready (informer cache has not synced), or the PodMonitor is missing. Static user login, UserAccount, expire, and provider checks will not run.

Check:

```
d8 k -n d8-user-authn get pods -l app=user-authn-controller
d8 k -n d8-user-authn describe pods -l app=user-authn-controller
```

- D8UserAuthnControllerUnavailable CE S6
  ▼ **user-authn-controller has unavailable replicas.**

  Deckhouse has detected unavailable replicas in the `user-authn-controller` deployment in the `d8-user-authn` namespace.

  While this alert is active, static Users may not get Dex Password objects, UserAccount projections stall, expired Users are not deleted, and DexProviderCheck / UserOperation stay pending.

  Check deployment and pod states:

  ```
  d8 k -n d8-user-authn get deploy user-authn-controller
  d8 k -n d8-user-authn get pods -l app=user-authn-controller
  d8 k -n d8-user-authn logs deploy/user-authn-controller --all-pods=true
  ```

- D8UserAuthnDexClientControllerTargetDown CE S6
  ▼ **Prometheus is unable to scrape dexclient-controller.**

  Prometheus cannot scrape metrics from `dexclient-controller` in namespace `d8-user-authn`.

  The controller is down, not Ready (informer cache has not synced), or the PodMonitor is missing. The credentials of DexClients are not kept up to date.

  Check:

  ```
  d8 k -n d8-user-authn get pods -l app=dexclient-controller
  d8 k -n d8-user-authn describe pods -l app=dexclient-controller
  ```

- D8UserAuthnDexClientControllerUnavailable CE S6
  ▼ **dexclient-controller has unavailable replicas.**

  Deckhouse has detected unavailable replicas in the `dexclient-controller` deployment in the `d8-user-authn` namespace.

  While this alert is active, a new DexClient gets no credentials Secret and no Dex client, and changes of DexClients are not applied.

  Check deployment and pod states:

  ```
  d8 k -n d8-user-authn get deploy dexclient-controller
  d8 k -n d8-user-authn get pods -l app=dexclient-controller
  d8 k -n d8-user-authn logs deploy/dexclient-controller --all-pods=true
  ```

- D8UserAuthnDexProviderTokenExchangeDeprecated CE S8
  ▼ **Tokens are exchanged through DexProvider XXX, which is deprecated.**

  In the last 24 hours, Dex exchanged tokens (RFC 8693 token exchange) through the DexProvider `XXX`. Token exchange through a DexProvider is deprecated. A DexProvider is meant for sign-in, and its connector does not check the audience of the token it exchanges.

  Move the exchange to a TokenExchangeProvider:

  1. Create a TokenExchangeProvider with the `issuer` of the DexProvider, the audience the exchanged tokens carry, and the DexClient that exchanges them in `spec.clients`.
  2. Set `spec.identity.username` to the claim the DexProvider took the user name from, for example `claim: email`, with a prefix such as `ci:`; the prefix is required.
  3. Before the switch, add the prefixed subjects to the ClusterAuthorizationRule, AuthorizationRule and RoleBinding objects that grant rights to the old names.
  4. In the exchange requests, replace `connector_id` with `status.connectorID` of the new object.
  5. After the switch, remove the old subjects from these objects.
  6. Turn off token exchange through the DexProvider by setting `spec.tokenExchange.enabled:`

`false`. It cannot be turned on again, and token exchange through a DexProvider will be removed after the next LTS release.

Find the clients that exchange tokens in the Dex log:

```
d8 k -n d8-user-authn logs deploy/dex -c dex --all-pods=true --since=24h | grep 'token exchange successful' | grep '"connector_id":"XXX"'
```

The alert stops 24 hours after the last exchange through the DexProvider.

- D8UserAuthnIdentityMigrationIncomplete CE S6
  ▼ Dex user data is not fully copied into UserIdentity objects.

  For more than 30 minutes, `user-authn-controller` has not finished copying the 2FA (TOTP) secret and the lock of Dex users from OfflineSessions and Password objects into UserIdentity objects.

  Until the copy of a provider and user pair completes, its 2FA secret and lock are stored only in the OfflineSessions and Password objects.

  The controller retries failed pairs by itself. Find the reason in its logs:

  ```
  d8 k -n d8-user-authn logs deploy/user-authn-controller --all-pods=true | grep -E 'identitymigration|UserIdentity'
  ```

  Typical reasons are API server errors and missing rights on `useridentities.dex.coreos.com`. Pairs that cannot be copied at all, for example an OfflineSessions object that cannot be decoded, are logged as `Not copying the pair into UserIdentity` and do not raise this alert.

- D8UserAuthnIDTokenTTLTooLong CE S6
  ▼ user-authn idTokenTTL is XXX, which is too long.

  The ModuleConfig `user-authn` sets `idTokenTTL: XXX`.

  Dex rotates the key that signs ID tokens every 6 hours (`signer.config.keysRotationPeriod: 6h`) and keeps a previous key only while tokens signed with it can still be valid.

  `idTokenTTL` maps to Dex `expiry.idTokens`. If that lifetime is greater than or equal to the signing-key rotation period (`6h`), every rotation must retain an extra previous key until all tokens from that key expire. Over time the set of retained signing keys grows without bound and Dex persists that growing key material.

  That is why `idTokenTTL` must be strictly less than `6h`. Lower it, for example:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: user-authn
  spec:
    version: 3
    settings:
      idTokenTTL: 1h
  ```

  Settings version 3 rejects `idTokenTTL >= 6h`. A ModuleConfig of an earlier settings version with such a value is still accepted and its value is converted to `5h59m`, so Dex already runs with `5h59m`; this alert keeps firing until the ModuleConfig itself is updated.

- D8UserAuthnRevocationFeedLagging CE S6
  ▼ The revocation feed of user-authn-controller is not delivering to dex-authenticator.

  For more than 5 minutes, a stream of the revocation feed of `user-authn-controller` has not written to its dex-authenticator for over 30 seconds. A stream writes at least a heartbeat every 10 seconds, so the stream is stuck.

  While the stream is stuck, a session that is revoked (user lock, password reset, logout) keeps working in that dex-authenticator until its next refresh in Dex: up to 30 seconds while Dex is reachable, and no longer than `idTokenTTL` otherwise.

  Check the logs of the controller and of the dex-authenticator pods:

```
d8 k -n d8-user-authn logs deploy/user-authn-controller --all-pods=true | grep revocation-feed
```

The controller closes a stream that falls behind, and the dex-authenticator reconnects by itself. A lag that does not go away usually means a network problem between the dex-authenticator pods and the controller pods.

- D8UserAuthnRevocationFeedNotConsumed CE S6
  ▼ **Pods of the shared dex-authenticator XXX do not receive revocations.**

  For more than 15 minutes, some pods of the shared dex-authenticator `XXX` have not received revocations from `user-authn-controller`. On those pods a revoked session stops working only at its next refresh in Dex.

  Check the logs of the pods and the network policies in the namespace `d8-user-authn`:

```
d8 k -n d8-user-authn logs -l app=shared-dex-authenticator -c dex-authenticator --prefix --
tail=100
```

Module user-authz

- D8UserAuthzAuthorizationRuleInvalid CE S7
  ▼ **The bindings of XXX XXXXXX/XXXNAME cannot be applied (XXX).**

  `user-authz-controller` cannot apply the bindings of XXX `XXXXXX/XXXNAME`; the `Ready` condition of the object is `False` with the reason `XXX`.

  - `InvalidSpec`: the rule cannot be rendered into bindings (unsupported `accessLevel`, an `accessLevel` not allowed for a namespaced rule, an `additionalRoles` entry that is not a ClusterRole or has an invalid name). Fix the object.
  - `ApplyError`: the API server rejected a write. The message of the condition names the binding and the error; a binding whose `roleRef` was changed by hand has to be deleted so that the controller recreates it.

```
d8 k XXX-n XXX XXXget XXX NAME -o jsonpath='{.status.conditions}'
```

- D8UserAuthzAuthorizationRulesInvalid CE S7
  ▼ **VALUE XXX objects cannot be applied (XXX).**

  `user-authz-controller` reports VALUE XXX objects whose bindings cannot be applied, reason `XXX`. Up to 50 of them are named by the `D8UserAuthzAuthorizationRuleInvalid` alerts; a large number with the reason `ApplyError` points at a cluster-wide cause (an admission webhook, API server errors) rather than at the objects.

```
d8 k get XXX -A -o custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,READY:'.status.conditions[?
(@.type=="Ready")].status',REASON:'.status.conditions[?(@.type=="Ready")].reason'
| grep -v True
d8 k -n d8-user-authz logs deploy/user-authz-controller --all-pods=true
```

- D8UserAuthzBindingsDrift CE S7
  ▼ **user-authz-controller cannot converge the bindings of XXX.**

  For more than 15 minutes some bindings of `XXX` objects differ from the state `user-authz-controller` wants (missing, superfluous, or with different subjects, roleRef, labels or owner). The controller reports drift only while its writes fail, so users may be missing access or keep access they should have lost.

  Find the objects and the reason:

```
d8 k get clusterauthorizationrules -o custom-
columns=NAME:.metadata.name,READY:'.status.conditions[?
(@.type=="Ready")].status',REASON:'.status.conditions[?
(@.type=="Ready")].reason',MESSAGE:'.status.conditions[?
(@.type=="Ready")].message'
d8 k get authorizationrules -A -o custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,READY:'.status.conditions[?
(@.type=="Ready")].status',MESSAGE:'.status.conditions[?
(@.type=="Ready")].message'
d8 k -n d8-user-authz logs deploy/user-authz-controller --all-pods=true
```

- D8UserAuthzControllerReconcileErrorsHigh CE S8
  ▼ **user-authz-controller reconcile errors are high for CONTROLLER_NAME.**

  Controller `CONTROLLER_NAME` of `user-authz-controller` has a reconcile error rate above 0.1 errors per second for more than 10 minutes.

  Check logs:

  ```
  d8 k -n d8-user-authz logs deploy/user-authz-controller --all-pods=true
  ```

- D8UserAuthzControllerTargetDown CE S6
  ▼ **Prometheus is unable to scrape user-authz-controller.**

  Prometheus cannot scrape metrics from `user-authz-controller` in the `d8-user-authz` namespace.

  The controller is down, not Ready (its informer cache has not synced), or the PodMonitor is missing. Access rules are not reconciled while it is down.

  Check:

  ```
  d8 k -n d8-user-authz get pods -l app=user-authz-controller
  d8 k -n d8-user-authz describe pods -l app=user-authz-controller
  d8 k -n d8-monitoring get podmonitor user-authz-controller
  ```

- D8UserAuthzControllerUnavailable CE S6
  ▼ **user-authz-controller has unavailable replicas.**

  Deckhouse has detected unavailable replicas in the `user-authz-controller` deployment in the `d8-user-authz` namespace.

  While this alert is active, changes to ClusterAuthorizationRule and AuthorizationRule objects are not applied to their ClusterRoleBindings and RoleBindings, and the `d8:dict` bindings and the RoleBindings of system and subsystem roles in module namespaces are not updated. Existing bindings keep working.

  Check deployment and pod states:

  ```
  d8 k -n d8-user-authz get deploy user-authz-controller
  d8 k -n d8-user-authz get pods -l app=user-authz-controller
  d8 k -n d8-user-authz logs deploy/user-authz-controller --all-pods=true
  ```

- D8UserAuthzCustomRolesNotAggregated CE S6
  ▼ **The aggregated ClusterRole of the XXX access level has no rules although custom roles for it exist.**

  Custom ClusterRoles annotated with `user-authz.deckhouse.io/access-level` are granted to users through the aggregated ClusterRole `user-authz:level:custom`, which `kube-controller-manager` fills from the roles labelled with the level. For more than 10 minutes the aggregated role of the `XXX` level has no rules while custom roles it must aggregate exist, so users of that level (and every level above it) have none of the custom permissions.

  Check that the aggregated role exists and that the `clusterrole-aggregation` controller of `kube-controller-manager` is running:

```
d8 k get clusterrole -l user-authz.deckhouse.io/access-level
d8 k get clusterrole -o name | grep 'user-authz:.*:custom' | xargs -n1 d8 k get -o yaml
d8 k -n kube-system logs -l component=kube-controller-manager --tail=200 | grep -i aggregat
```

- D8UserAuthzDeprecatedRBACv2CapabilityInUse CE S5
  ▾ A binding references a deprecated RBACv2 capability that no longer grants access.

  The `XXX` `XXX` XXX in namespace `NAMESPACE` XXX binds the deprecated RBACv2 capability `XXX` directly.

  Capabilities (`d8:manage:permission:*`, `d8:use:capability:*`) are aggregation building blocks — they are meant to be composed into the `d8:{system,subsystem,namespace,project}:*` roles, not bound directly. They were renamed and have NO compatibility alias, so this binding **already grants nothing** after the upgrade.

  No ClusterRole named `XXX` exists in the cluster. The alert also names a binding to an old role name that a module no longer ships, for example after the module moved to the new role names.

  Recreate the binding against the appropriate new role (or a custom role aggregating the new capability). Because `roleRef` is immutable, delete and recreate the binding:

  ```
  d8 k get XXX XXXXXX -n NAMESPACEXXX -o yaml
  ```

- D8UserAuthzDeprecatedRBACv2ModuleRoleInUse CE S7
  ▾ A binding references the old name of a role a module renamed.

  The `XXX` `XXX` XXX in namespace `NAMESPACE` XXX references `XXX`, the old name of a role that a module renamed to `XXX`.

  The module keeps the old name as a compatibility alias, so the binding keeps working. The alias may grant less than `XXX`, for example when the module moved into a wider subsystem and the alias kept the rights of the old role, so check what the subjects of the binding need before recreating it. The module removes the alias in one of its own releases; the Deckhouse release is not held back by this binding. See the documentation of the module for when.

  Recreate the binding on the new name. Because `roleRef` is immutable, delete and recreate the binding pointing at `XXX`:

  ```
  d8 k get XXX XXXXXX -n NAMESPACEXXX -o yaml
  ```

- D8UserAuthzDeprecatedRBACv2RoleInUse CE S7
  ▾ A binding still references a deprecated RBACv2 role.

  The `XXX` `XXX` XXX in namespace `NAMESPACE` XXX references the deprecated RBACv2 ClusterRole `XXX`.

  The RBACv2 role model was renamed: `d8:manage:subsystem|all:level` became `d8:system:*` / `d8:subsystem:*:*`, and `d8:use:role:level` became `d8:namespace:level`. This name still works via a compatibility alias kept for one release and will be removed in the next one — after that this binding will grant nothing. The release that removes the aliases is held back by the `deprecatedRBACv2BindingsCount` release requirement until this binding is recreated on the new name.

  Migrate the binding to the new role name. Because `roleRef` is immutable, delete and recreate the binding pointing at the new role:

  ```
  d8 k get XXX XXXXXX -n NAMESPACEXXX -o yaml
  # Recreate it with roleRef.name set to the new role (d8:system:*,
  d8:subsystem:*, or d8:namespace:*).
  ```

  XXX `user-authz-controller` renders this binding from `spec.additionalRoles` of the ClusterAuthorizationRule `XXX` and creates it again when it is deleted. Replace `XXX` with the new role name in `spec.additionalRoles` of that ClusterAuthorizationRule instead.XXXIf a project template renders this binding, replace the role name in the ProjectTemplate instead, because `multitenancy-manager` renders the binding again from it.XXXXXXXXX

  In the new model the rights for `user-authn` and `user-authz` XXX and the right to assign rolesXXX come with `d8:subsystem:iam:XXX`, not with `d8:subsystem:security:XXX`. To keep them, replace this binding with two bindings with the same subjects, one to `d8:subsystem:iam:XXX` and one to `d8:subsystem:security:XXX`. `d8:subsystem:iam:XXX` also grants the rights for `multitenancy-manager`.XXX

The list of subsystems changed as well. `deckhouse`, `infrastructure` and `kubernetes` were merged into `cluster`, `networking` became `network`, the `user-authz` and `user-authn` modules moved from `security` to `iam`, and `multitenancy-manager` moved from `deckhouse` to `iam`. The aliases of the old subsystem roles select the capabilities of the modules these roles covered, and the old subsystem labels that modules released on their own may still carry. The capabilities themselves changed in this release, so the rights of the binding are close to the ones it had before the upgrade, not the same. The replacement `d8:subsystem:cluster:level` covers the whole `cluster` subsystem and grants more, and so do the roles of the other subsystems that the description of the alias names, so check which roles the subjects of the binding need. The user-authz documentation lists the remaining differences.

A ClusterRoleBinding to a deprecated `d8:manage:*` name also keeps the RoleBindings `d8:use:use role:binding:name` that the previous release created for it in the namespaces of the modules of its old subsystem (`d8-cloud-provider-*`, `d8-monitoring` and the rest). The use role is `viewer` for a `viewer` alias and `user` for a `manager` alias. When the binding is recreated on the new role name, they are replaced by RoleBindings to the namespace role of the new role in the namespaces of the modules of the new subsystem.

Note: the new `d8:namespace:admin` no longer grants `serviceaccounts/token` creation or impersonation — these disaster-grade actions now require a `superadmin` role by design.

- D8UserAuthzForeignAggregationLabel CE S6
  ▾ ClusterRole NAME carries the XXX aggregation label without being a custom capability.

  The ClusterRole `NAME` carries the `rbac.deckhouse.io/aggregate-to-XXX-as` label but is not labelled `rbac.deckhouse.io/kind: custom-capability`XXX (its kind label is `XXX`)XXX. The aggregation controller reads nothing but that label, so it merges the rules of this role into XXXthe platform `d8:XXX:*` rolesXXXthe deprecated `d8:manage:all:*` and `d8:use:role:*` roles and every custom role that selects the labelXXXthe `d8:subsystem:XXX:*` roles, if a subsystem has this name, and into every custom role that selects the labelXXX. Every holder of these roles in the cluster gets the rules, outside the role model’s contract.

  For the aggregation label of the system scope, of the namespace or project scope, or of a built-in subsystem, the `rbacv2-cluster-roles.deckhouse.io` webhook refuses such an object, and every `UPDATE` of this role, including a GitOps re-apply, is denied until it is fixed. Choose one:

  - If the role is meant to extend the platform roles, replace it with a custom capability. The name of an object cannot be changed, so create `d8:custom:scope-capability:name` with the same rules and aggregation label, the `rbac.deckhouse.io/kind: custom-capability` and `rbac.deckhouse.io/scope: scope` labels, and no `aggregationRule`, and then delete this role. Deckhouse reads `rbac.deckhouse.io/namespace` only on the ClusterRoles it ships: if this role has that label, neither it nor its replacement gives RoleBinding objects in that namespace, so bind the subjects to a namespace role there with a RoleBinding. A capability of the old scheme with the `rbac.deckhouse.io/kind: use` label and `rbac.deckhouse.io/aggregate-to-kubernetes-as: LEVEL` or `rbac.deckhouse.io/aggregate-to-all-as: LEVEL` extended the namespace roles, so its replacement is `d8:custom:namespace-capability:name` with `rbac.deckhouse.io/scope: namespace` and `rbac.deckhouse.io/aggregate-to-namespace-as: LEVEL` instead. For the commands, refer to the corresponding section in the `user-authz` module FAQ.
  - Otherwise remove the `rbac.deckhouse.io/aggregate-to-*-as` labels from it.

  List every such role:

  ```
  d8 k get clusterroles -o json | jq -r '.items[] | select((.metadata.labels // {}) |
  to_entries | any(.key | startswith("rbac.deckhouse.io/aggregate-to-"))) |
  select((.metadata.labels.heritage // "") != "deckhouse") |
  select((.metadata.labels["rbac.deckhouse.io/kind"] // "") != "custom-
  capability") | .metadata.name'
  ```

- D8UserAuthzLegacyRBACv2CustomRoleFound CE S7
  ▾ The custom role NAME has lost access it gave through the labels of an earlier role model.

  The ClusterRole `NAME` has no `heritage: deckhouse` label and has an `aggregationRule` field, and one of the following holds:

  - A selector of its `aggregationRule` field requires the `rbac.deckhouse.io/kind: manage` or `use` label of the experimental role model. The built-in capabilities no longer carry these labels, so such a selector matches none of them. The Kubernetes aggregation controller keeps from it only the rules of the custom capabilities that still carry the old labels, and the subjects of the role have lost the other permissions it gave. Custom roles get no compatibility aliases.

- The role has the `rbac.deckhouse.io/use-role` label without `rbac.deckhouse.io/scope: system` or `subsystem`. Deckhouse creates the RoleBinding objects for the subjects of a role in the namespaces of the modules only for roles with that scope, so it has deleted the ones it had created for this role, even if its selectors are already rewritten.
- A selector of its `aggregationRule` field requires the aggregation label of a lineage that no built-in role collects, such as `rbac.deckhouse.io/aggregate-to-all-as` of the old scheme, the label of a subsystem the role model no longer has, or the label of a subsystem a module has left. The modules label their capabilities only with the lineages that the built-in roles collect, so such a selector matches none of the built-in capabilities either. A selector by a lineage that a custom capability carries is not counted, because it still gets the rules of that capability. A custom capability here is a ClusterRole without the `heritage: deckhouse` label and without an `aggregationRule` field. Neither is a selector by the lineage of the subsystem in the `rbac.deckhouse.io/subsystem` label of the role, because the custom capabilities of that subsystem carry it. Neither exception applies to the `all`, `deckhouse`, `infrastructure`, `kubernetes` and `networking` lineages, which do not count as custom ones. A compatibility alias of an old role, a ClusterRole with the `rbac.deckhouse.io/deprecated: "true"` label, does not count as a built-in role that collects a lineage, because it collects the lineage of an old subsystem only for the bindings to the old name.

Repair the access in this order:

1. Create a role of the new scheme next to this one. Give it the `d8:custom:` name prefix and the `rbac.deckhouse.io/kind: custom-role` label, and select the capabilities by the labels of the new scheme, without `rbac.deckhouse.io/kind`. Select the capabilities that a selector by a lineage no built-in role collects was meant for by their `rbac.deckhouse.io/capability` labels, or by `rbac.deckhouse.io/scope: system` and the `module` label of their module. If this role has the `rbac.deckhouse.io/use-role` label, keep the label on the new role and add `rbac.deckhouse.io/scope: subsystem` with `rbac.deckhouse.io/subsystem: SUBSYSTEM`, or `rbac.deckhouse.io/scope: system`, so that Deckhouse creates the RoleBinding objects in the namespaces of the modules again.
2. Create capabilities of the new scheme in place of the custom capabilities of the old scheme that this role aggregates. The `D8UserAuthzForeignAggregationLabel` alert names them.
3. For every RoleBinding and ClusterRoleBinding object to this role, create one with the same subjects to the new role. If this role has the `rbac.deckhouse.io/aggregate-to-all-as: LEVEL` label, what it extended depends on its kind label. With `rbac.deckhouse.io/kind: manage`, its permissions also reached the subjects of `d8:manage:all:LEVEL`. A custom role cannot be aggregated into `d8:system:LEVEL`, so bind the new role to those subjects with a ClusterRoleBinding. With `rbac.deckhouse.io/kind: use`, they reached the subjects of `d8:use:role:LEVEL` in every namespace, so move the permissions into a namespace capability with `rbac.deckhouse.io/aggregate-to-namespace-as: LEVEL`, which extends `d8:namespace:LEVEL`.
4. Check the permissions of the subjects with `d8 k auth can-i VERB RESOURCE --as=USER_NAME`.
5. Delete the old bindings, this role and the old capabilities.

For the commands, refer to the corresponding section in the `user-authz` module FAQ.

List all such roles:

```
d8 k get clusterroles -o json | jq -r '
  def lineages: [((.matchLabels // {}) | keys[]), (.matchExpressions[]? |
select(.operator == "In" or .operator == "Exists") | .key)]
    | map(capture("^rbac\\.deckhouse\\.io/aggregate-to-(?lineage.+)-
as$").lineage);
  ([.items[] | select(.metadata.labels.heritage == "deckhouse" and
.metadata.labels["rbac.deckhouse.io/deprecated"] != "true")
        | .aggregationRule.clusterRoleSelectors[]? | lineages[]] + ["system",
"namespace", "project"]) as $collected
  | ([.items[] | select(.metadata.labels.heritage != "deckhouse" and
.aggregationRule == null) | (.metadata.labels // {}) | keys[]
        | capture("^rbac\\.deckhouse\\.io/aggregate-to-(?lineage.+)-
as$").lineage]) as $carried
  | .items[]
  | select(.metadata.labels.heritage != "deckhouse" and .aggregationRule !=
null)
  | (.metadata.labels["rbac.deckhouse.io/subsystem"] // "") as $own
  | select(
      (
        ((.metadata.labels["rbac.deckhouse.io/use-role"] // "") != "")
        and (((.metadata.labels["rbac.deckhouse.io/scope"] // "") |
IN("system", "subsystem")) | not)
      )
      or (
        [
          .aggregationRule.clusterRoleSelectors[]?
          | (.matchLabels["rbac.deckhouse.io/kind"] // empty),
            (.matchExpressions[]? | select(.key == "rbac.deckhouse.io/kind" and
.operator == "In") | .values[])
        ]
        | any(IN("manage", "use"))
      )
      or (
        [
          .aggregationRule.clusterRoleSelectors[]? | lineages[]
          | select((IN($collected[]) | not)
              and (IN("all", "deckhouse", "infrastructure", "kubernetes",
"networking") or (. != $own and (IN($carried[]) | not))))
        ]
        | length > 0
      )
    )
  | .metadata.name'
```

- D8UserAuthzLegacyUseCapability CE S7
  ▼ ClusterRole NAME carries the use labels of the previous scheme, which the namespace and project roles do not collect.

  The ClusterRole `NAME`XXX of the `MODULE NAME` moduleXXX carries the `rbac.deckhouse.io/kind: use` and `rbac.deckhouse.io/aggregate-to-XXX-as: XXX` labelsXXX of the use roles that a module with a subsystem of its own shipped in the previous schemeXXX of a use capability of the previous schemeXXX. `rbac.deckhouse.io/kind: use` is the label of a use capability of the previous schemeXXX. XXXThe moduleXXXThe capabilityXXX is still on that scheme.

  XXXThe `d8:namespace:*` and `d8:project:*` roles do not collect this label. Only the deprecated `d8:use:role:*` aliases still collect it, and they will be removed in the next release. A RoleBinding moved from `d8:use:role:XXX` to `d8:namespace:XXX` and a project role therefore do not get the rules of this ClusterRole.XXXNo role collects a ClusterRole with `rbac.deckhouse.io/kind: use` and this label. The `d8:system:*` and `d8:subsystem:*` roles and the deprecated `d8:manage:*` aliases leave out `rbac.deckhouse.io/kind: use`, and the `d8:namespace:*`, `d8:project:*` and `d8:use:role:*` roles do not select this label. Only a binding to this ClusterRole itself gives its rules.XXX

  Choose one:

- If the ClusterRole belongs to a module, update the module to a release that labels its namespace capabilities with `rbac.deckhouse.io/kind: capability` and `rbac.deckhouse.io/aggregate-to-namespace-as`. Until then, give the subjects that need these rules a RoleBinding to `NAME` in their namespace, next to their RoleBinding to the `d8:namespace:*` role of their level. Neither role is a deprecated alias, so these RoleBinding objects do not hold the next release. In a project namespace, the `cluster-objects-grants-validator.multitenancy.deckhouse.io` webhook admits a RoleBinding to `NAME` only if the ClusterRole has the `rbac.deckhouse.io/delegatable: "true"` labelXXX, which the use capabilities of the previous scheme do not haveXXX.XXX Where the webhook refuses it, keep the RoleBinding to the `d8:use:role:*` alias: it keeps these rules, but the `D8UserAuthzDeprecatedRBACv2RoleInUse` alert reports it, and the next release, which removes the aliases, stays pending until the module is updated and the RoleBinding is moved to the `d8:namespace:*` role.XXX Where the webhook refuses it, the subjects get these rules only once the module is updated.XXX
- If it is a custom capability, replace it with `d8:custom:namespace-capability:NAME` with the same rules and the `rbac.deckhouse.io/kind: custom-capability`, `rbac.deckhouse.io/scope: namespace` and `rbac.deckhouse.io/aggregate-to-namespace-as: XXX` labels. For the commands, refer to the corresponding section in the `user-authz` module FAQ.

List every such ClusterRole:

```
d8 k get clusterroles -l rbac.deckhouse.io/kind=use -o json | jq -r '.items[] |
select(.metadata.labels | keys | map(capture("^rbac\\.deckhouse\\.io/aggregate-
to-(?lineage.+)-as$").lineage) | any(IN("kubernetes", "all", "system", "iam",
"security", "cluster", "delivery", "network", "storage", "observability",
"managed-services", "deckhouse", "infrastructure", "networking"))) |
.metadata.name'
```

- D8UserAuthzPermissionBrowserUnavailable CE S6
  ▾ Permission Browser API server has unavailable replicas.

  Deckhouse has detected unavailable replicas in the `permission-browser-apiserver` deployment in the `d8-user-authz` namespace. Permission Browser requests can fail while this alert is active.

  Check deployment and pod states:

  ```
  d8 k -n d8-user-authz get deploy permission-browser-apiserver
  d8 k -n d8-user-authz get pods -l app=permission-browser-apiserver
  ```

- D8UserAuthzReleaseSlow CE S6
  ▾ The user-authz module release repeatedly takes more than 5 minutes.

  Applying the `user-authz` Helm release took more than 5 minutes on average over the last hour, and it was applied at least twice in that hour, so this is not the single slow pass of a cluster bootstrap or of the one-time migration to `user-authz-controller`. A release that does not finish within the module release timeout (20 minutes) fails, changes to ClusterAuthorizationRule and AuthorizationRule objects stop being applied and platform updates stall (the `D8DeckhouseCouldNotRunModule` alert fires in that case).

  The release itself no longer grows with the number of rules: bindings are managed by `user-authz-controller`, and custom ClusterRoles are aggregated. A slow release therefore points outside the module: an API server that answers slowly, an object stuck with a finalizer, a Deckhouse Pod short of CPU, or the release engine itself. Check what the release is waiting on:

  ```
  d8 s queue list
  d8 k -n d8-system logs deploy/deckhouse | grep '"module":"user-authz"'
  ```

  If the cluster has just been updated to the release that moved the bindings to the controller, one slow pass is expected and does not raise this alert by itself; if it fires anyway, the pass is being retried. It is proportional to the number of bindings and can exceed the timeout on clusters with more than 5000 of them; the module FAQ describes what to do in that case.

- D8UserAuthzRenderSlow CE S7
  ▾ Rendering the user-authz module templates repeatedly takes more than 30 seconds.

  Rendering the templates of the `user-authz` module took more than 30 seconds on average over the last 30 minutes, across at least three renders; normally one render takes about a second. Rendering is CPU work inside the Deckhouse Pod and does not depend on the number of ClusterAuthorizationRule or AuthorizationRule objects, so the cause is the node, not the module: most often the Deckhouse Pod is short of CPU (CPU steal on a virtual machine, a node running at

its limits, a bootstrap that starts every module at once).

A single slow render during a cluster bootstrap does not raise this alert by itself. If it fires, compare the module with the others and look at the node:

```
topk(10, increase(deckhouse_helm_operation_seconds_sum{operation="template"}[30m]) /
increase(deckhouse_helm_operation_seconds_count{operation="template"}[30m]))
rate(node_cpu_seconds_total{mode="steal"}[5m])
```

```
d8 k -n d8-system logs deploy/deckhouse | grep '"module":"user-authz"'
```

- D8UserAuthzRetiredAggregationLabel CE S7
  ▾ ClusterRole NAME aggregates by the label of the retired XXX subsystem.

  The ClusterRole `NAME`XXX of the `MODULE_NAME` moduleXXX carries the `rbac.deckhouse.io/aggregate-to-XXX-as` label.

  The role model has no `XXX` subsystem any more. `deckhouse`, `infrastructure` and `kubernetes` were merged into `cluster`, and `networking` was renamed to `network`. No `d8:system:*` or `d8:subsystem:*` role collects this label, so the holders of these roles do not get the rules of this ClusterRole. Only the deprecated `d8:manage:*` aliases still collect it, and they will be removed in the next release.

  Choose one:

  - If the ClusterRole belongs to a module, update the module to a release that labels its capabilities with the lineage of its new subsystem. For most modules of the old subsystem that is `rbac.deckhouse.io/aggregate-to-XXX-as`. The subsystem of each module is listed in the user-authz documentation.
  - If it is a custom capability, replace the label with `rbac.deckhouse.io/aggregate-to-XXX-as` or with the label of the subsystem the capability belongs to now.

  Until then, a subject bound to a new role gets the rules of such a capability of a module only through a custom role. Create a `d8:custom:system:NAME` ClusterRole with the `rbac.deckhouse.io/kind: custom-role` and `rbac.deckhouse.io/scope: system` labels and bind it to the subjects with a ClusterRoleBinding. A capability of the previous scheme has no `rbac.deckhouse.io/capability` label, so the role selects the capabilities of the module by the `module: XXXMODULE_NAMEXXXMODULEXXX` and `rbac.deckhouse.io/kind: manage` labels. Such a selector takes the capabilities of every level, so add `rbac.deckhouse.io/aggregate-to-XXX-as: viewer` to it for a role that only views. The `D8UserAuthzLegacyRBACv2CustomRoleFound` alert names this role, because it selects by a label of the previous scheme. Once the module carries the new labels, select its capabilities by their `rbac.deckhouse.io/capability` label instead. A capability that already carries `rbac.deckhouse.io/kind: capability` is selected by its `rbac.deckhouse.io/capability` label right away.

  List every such ClusterRole:

  ```
  d8 k get clusterroles -o json | jq -r '.items[] |
  select((.metadata.labels["rbac.deckhouse.io/kind"] // "") != "use") |
  select((.metadata.labels // {}) | keys | any(IN("rbac.deckhouse.io/aggregate-
  to-deckhouse-as", "rbac.deckhouse.io/aggregate-to-infrastructure-as",
  "rbac.deckhouse.io/aggregate-to-kubernetes-as", "rbac.deckhouse.io/aggregate-
  to-networking-as"))) | .metadata.name'
  ```

- D8UserAuthzRevocationCheckNotSynced CE S6
  ▾ The revocation check of the user-authz webhook has not loaded the revocations.

  For more than 10 minutes the `user-authz-webhook` instance `POD NAME` has not loaded the `Revocation` objects of the `user-authn` module (the objects in `d8-user-authn` that record revoked user credentials): it cannot list them, or cannot find out whether the API server serves them.

  Until it loads them, the instance has no opinion about revoked credentials: on the master node of this instance, `kubectl` with the token of a locked or logged-out user keeps working until the token expires. Other authorization decisions are not affected.

  Check the webhook's access to the API and its logs:

  ```
  d8 k -n d8-user-authz logs POD_NAME -c webhook --tail=100 | grep -i revocation
  d8 k auth can-i list revocations.dex.coreos.com -n d8-user-authn --as=system:serviceaccount:d8-
  user-authz:webhook
  ```

- D8UserAuthzRevocationCheckStale CE S6
  ▾ **The revocation check of the user-authz webhook no longer sees new revocations.**

  For more than 15 minutes the `user-authz-webhook` instance `POD_NAME` has been unable to watch the `Revocation` objects of the `user-authn` module (the objects in `d8-user-authn` that record revoked user credentials).

  The instance keeps refusing the Dex tokens it already knows to be revoked, but a revocation made since its watch broke does not reach it: on the master node of this instance, `kubectl` with the token of a locked or logged-out user keeps working until the token expires.

  Check the webhook's access to the API and its logs:

  ```
  d8 k -n d8-user-authz logs POD_NAME -c webhook --tail=100 | grep -i revocation
  d8 k auth can-i watch revocations.dex.coreos.com -n d8-user-authn --as=system:serviceaccount:d8-user-authz:webhook
  ```

- D8UserAuthzRuleNeedsMultiTenancy CE S6
  ▾ **A ClusterAuthorizationRule asks for namespace limits that nothing enforces.**

  The `ClusterAuthorizationRule` `NAME` sets `XXX`, but the `enableMultiTenancy` setting of the `user-authz` module is off, so the authorization webhook that enforces those options is not deployed.

  The rule is not partially applied. It is applied without its limits: the subjects named in it hold its access level in **every** namespace of the cluster, the system ones included.

  Either enable multi-tenancy, or remove the options from the rule:

  ```
  d8 k get clusterauthorizationrule NAME -o yaml
  d8 k get moduleconfig user-authz -o jsonpath='{.spec.settings.enableMultiTenancy}'
  ```

- D8UserAuthzRulePropagationLag CE S6
  ▾ **One user-authz webhook instance stopped picking up rule changes.**

  One instance of `user-authz-webhook` has not updated its rules for more than an hour while another instance has. The rules are changing in the cluster and this instance does not see the changes.

  Find the instance and check its log:

  ```
  d8 k -n d8-user-authz get pods -l app=user-authz-webhook -o wide
  d8 k -n d8-user-authz logs -l app=user-authz-webhook -c webhook --tail=100
  ```

  An instance whose watch keeps failing becomes not ready, so check the readiness of the Pod as well.

- D8UserAuthzRulesNeedMultiTenancy CE S6
  ▾ **More ClusterAuthorizationRules ask for namespace limits than the per-rule alert names.**

  `VALUE` ClusterAuthorizationRules set `limitNamespaces`, `namespaceSelector` or `allowAccessToSystemNamespaces` while the `enableMultiTenancy` setting of the `user-authz` module is off. `D8UserAuthzRuleNeedsMultiTenancy` names the first 50 of them; this alert accounts for the rest.

  None of those rules is partially applied: the subjects named in each of them hold its access level in **every** namespace of the cluster, the system ones included.

  Check whether multi-tenancy was meant to be enabled, and list the affected rules:

  ```
  d8 k get moduleconfig user-authz -o jsonpath='{.spec.settings.enableMultiTenancy}'
  d8 k get clusterauthorizationrule -o json | jq -r '.items[] |
  select((.spec.limitNamespaces // [] | length > 0) or (.spec.namespaceSelector
  != null) or (.spec.allowAccessToSystemNamespaces == true)) | .metadata.name'
  ```

- D8UserAuthzWebhookDirectoryDiverged CE S6
  ▾ **The masters disagree about the multi-tenancy rules.**

  Instances of `user-authz-webhook` have been using different sets of `ClusterAuthorizationRules` for 10 minutes.

Each master node authorizes the requests that reach it, so the same request is answered differently depending on the node: a user can be inside their namespace limits on one node and outside them on another.

Find the instance that is behind and look at why:

```
d8 k -n d8-user-authz get pods -l app=user-authz-webhook -o wide
d8 k -n d8-user-authz logs -l app=user-authz-webhook -c webhook --tail=100 | grep -i
'list/watch'
```

A growing `user authz webhook rules watch errors total` on one instance points at its watch; a `user_authz_webhook_rules_directory_updated_timestamp_seconds` that has stopped moving points at rules that are no longer updated. Deleting the lagging Pod is the last resort: the DaemonSet recreates it, and on a cluster with a single master node every request that the authorization webhook has to answer is denied while the Pod restarts. Control plane components, kubelets, and the ServiceAccounts of the `kube-system` and `d8-*` namespaces keep working, because the webhook does not check them.

- D8UserAuthzWebhookRulesQuarantined CE S7
  ▾ **A ClusterAuthorizationRule cannot be compiled by the webhook.**

  For more than 10 minutes the `user-authz-webhook` has quarantined at least one `ClusterAuthorizationRule`: its `limitNamespaces` pattern or `namespaceSelector` does not compile, or the rule cannot be read at all.

  A rule with a broken filter is applied without that filter, so its subjects get a narrower scope than written and are denied where the filter was meant to allow. A rule that cannot be read is not applied at all, and its subjects are denied every namespace.

  Find the rule:

  ```
  d8 k get clusterauthorizationrules -o custom-
  columns=NAME:.metadata.name,LIMIT:.spec.limitNamespaces,SELECTOR:.spec.namespaceSelector
  d8 k -n d8-user-authz logs -l app=user-authz-webhook -c webhook --tail=100 | grep -i quarantin
  ```

- D8UserAuthzWebhookRulesWatchErrors CE S7
  ▾ **The user-authz webhook cannot watch the ClusterAuthorizationRules.**

  For more than 10 minutes the `user-authz-webhook` rules informer has been failing to watch the `ClusterAuthorizationRules` (a non-zero rate of watch errors). New or changed rules may not be reaching the webhook, so multi-tenancy decisions can lag behind what users configure.

  Check the webhook’s access to the API and its logs:

  ```
  d8 k -n d8-user-authz logs -l app=user-authz-webhook -c webhook --tail=100
  d8 k auth can-i watch clusterauthorizationrules.deckhouse.io --as=system:serviceaccount:d8-user-
  authz:webhook
  ```

- D8UserAuthzWebhookTargetDown CE S6
  ▾ **Prometheus is unable to scrape the user-authz webhook.**

  Prometheus cannot collect metrics from at least one instance of `user-authz-webhook` in the `d8-user-authz` namespace. One instance without metrics is enough for the alert to fire.

  If the instance is running and only its metrics are unreachable, it keeps answering authorization requests, but the other alerts of the module cannot see its state. If it is not running or has not read the rules yet, it answers `503` to every authorization request, and the API server denies the request, because the webhook is configured with `failurePolicy: Deny`.

  Check the DaemonSet, its metrics sidecar and the scrape target:

  ```
  d8 k -n d8-user-authz get pods -l app=user-authz-webhook -o wide
  d8 k -n d8-user-authz logs -l app=user-authz-webhook -c kube-rbac-proxy --tail=50
  ```

Module vertical-pod-autoscaler

- VerticalPodAutoscalerDeprecatedUpdateModeAuto CE S9
  ▾ **VerticalPodAutoscaler objects use the deprecated Auto update mode.**

  There are `VALUE` VerticalPodAutoscaler object(s) in the cluster configured with the deprecated

update mode (`spec.updatePolicy.updateMode: Auto`).

The `Auto` update mode is deprecated. It currently behaves like `Recreate` (the VPA updater evicts pods to apply new resource requests) and its semantics are reserved to change in future versions, which makes cluster behavior ambiguous and potentially disruptive.

Set the affected VPA object(s) to one of the supported update modes (`spec.updatePolicy.updateMode`):

- `InPlaceOrRecreate`: Apply recommendations in place when possible, fall back to recreation (recommended).
- `Recreate`: Evict and recreate pods to apply recommendations.
- `Initial`: Apply only when pods are (re)created.
- `Off`: Recommendations only, no changes to running pods.

To list the affected objects, open the **VPA Deprecated Update Mode** Grafana dashboard, or run:

```
kube_verticalpodautoscaler_spec_updatepolicy_updatemode{update_mode="Auto"} == 1
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Sysctl parameters managed by the platform

Deckhouse automatically configures and manages a set of the server's kernel parameters using the `sysctl` utility. The configured parameters improve network throughput, prevent resource depletion, and optimize memory management.

If you modify these parameters, Deckhouse will automatically revert them to the values listed below.

Parameter	Value set by Deckhouse	Description
`/sys/block/*/queue/nr_requests`	`256`	Number of queued requests for block devices.
`/sys/block/*/queue/read_ahead_kb`	`256`	Amount of extra data that kernel reads from the disk to improve future read performance.
`/sys/kernel/mm/transparent_hugepage/enabled`	`never`	Disables Transparent HugePage.
`/sys/kernel/mm/transparent_hugepage/defrag`	`never`	Disables the Transparent HugePage defragmentation.
`/sys/kernel/mm/transparent_hugepage/use_zero_page`	`0`	Disables usage of huge zero pages.
`/sys/kernel/mm/transparent_hugepage/khugepaged/defrag`	`0`	Disables `khugepaged` defragmentation.
`/proc/sys/net/ipv4/conf/*/rp_filter`	`0`	Disables reverse path filtering for all interfaces.
`fs.file-max`	`1000000`	Maximum number of open files.
`fs.inotify.max_user_instances`	`5120`	Maximum number of inotify instances.
`fs.inotify.max_user_watches`	`524288`	Maximum number of files monitored by a single inotify instance.

Parameter	Value set by Deckhouse	Description
`fs.may_detach_mounts`	`1`	Allows lazy unmounting of a file system.
`kernel.numa_balancing`	`0`	Disables automatic NUMA memory balancing.
`kernel.panic`	`10 (0 if fencing is enabled)`	Time in seconds until the node reboots after it encounters the fatal kernel panic error. By default, it’s set to `10`. If `fencing` mode is enabled for the node, it’s set to `0`, preventing the node from rebooting.
`kernel.panic_on_oops`	`1`	Allows the system to trigger a kernel panic after an unexpected oops error. Required for kubelet to work correctly.
`kernel.pid_max`	`2000000`	Maximum number of process IDs that can be assigned in the system.
`net.bridge.bridge-nf-call-arptables`	`1`	Enables traffic filtering through arptables. Required for kube-proxy to work correctly.
`net.bridge.bridge-nf-call-ip6tables`	`1`	Enables traffic filtering through ip6tables. Required for kube-proxy to work correctly.
`net.bridge.bridge-nf-call-iptables`	`1`	Enables traffic filtering through iptables. Required for kube-proxy to work correctly.
`net.core.netdev_max_backlog`	`5000`	Maximum number of packets allowed in the processing queue.
`net.core.rmem_max`	`16777216`	Maximum receive buffer size in bytes.
`net.core.somaxconn`	`1000`	Maximum number of pending connections.
`net.core.wmem_max`	`16777216`	Maximum send buffer size in bytes.
`net.ipv4.conf.all.forwarding`	`1`	Enables IPv4 packet forwarding between network interfaces. Equivalent to the `net.ipv4.ip_forward` parameter.
`net.ipv4.ip_local_port_range`	`"32768 61000"`	Range of ports available for outgoing TCP and UDP connections.
`net.ipv4.neigh.default.gc_thresh1`	`16384`	Lower threshold for the amount of ARP entries after which the system starts cleaning up old entries.
`net.ipv4.neigh.default.gc_thresh2`	`28672`	Middle threshold for the amount of ARP entries after which the system starts garbage collection.
`net.ipv4.neigh.default.gc_thresh3`	`32768`	Absolute maximum number of ARP entries.

Parameter	Value set by Deckhouse	Description
`net.ipv4.tcp_max_syn_backlog`	`8096`	Maximum number of queued SYN connections.
`net.ipv4.tcp_no_metrics_save`	`1`	Disables saving of TCP metrics of closed connections and reusing them for new connections.
`net.ipv4.tcp_rmem`	`"4096 12582912 16777216"`	Receive buffer sizes for incoming TCP packets in bytes: `"<minimum> <default> <maximum>"`.
`net.ipv4.tcp_slow_start_after_idle`	`0`	Disables using the congestion window (CWND) and slow start algorithm for TCP connections.
`net.ipv4.tcp_tw_reuse`	`1`	Enables reusing the outgoing TCP connections in `TIME-WAIT` state.
`net.ipv4.tcp_wmem`	`"4096 12582912 16777216"`	Send buffer sizes for outgoing TCP packets in bytes: `"<minimum> <default> <maximum>"`.
`net.netfilter.nf_conntrack_max`	`<no-of-cores * 131072> or 524288`	Maximum number of tracked connections in the conntrack table. Calculated as “number of CPU cores” * 131072, but no lower than `524288`.
`net.nf_conntrack_max`	`<no-of-cores * 131072> or 524288`	Maximum number of tracked connections in the conntrack table for older kernels. Calculated as “number of CPU cores” * 131072, but no lower than `524288`.
`vm.dirty_background_ratio`	`5`	Percentage of system memory that can be filled with dirty pages before the kernel starts writing them to disk in the background.
`vm.dirty_expire_centisecs`	`12000`	Duration (in centiseconds) a dirty page can remain in system memory before it must be written to disk.
`vm.dirty_ratio`	`80`	Percentage of system memory that can be filled with dirty pages before all processes must stop and flush data to disk.
`vm.min_free_kbytes`	`131072`	Minimum amount of free memory in kilobytes reserved by the kernel for critical operations.
`vm.overcommit_memory`	`1`	Enables memory overcommitment.
`vm.swappiness`	`0`	Disables swap file usage.

Additional resources

- Module `node-manager` custom resources

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Information about used software

List of third-party software used in the Deckhouse Platform. Includes **only built-in modules** (doesn't include modules from sources).

The list of third-party software used in a specific module can be found on the overview page in the module documentation.

Amazon Web Services Cloud provider

Versions: v1.33.4, v1.34.3, v1.35.2, v1.36.0, v1.37.0

Cloud Controller Manager for Amazon Web Services.

Used in module cloud-provider-aws

Apache License 2.0

Amazon Web Services CSI driver

Versions: v1.54.0, v1.66.0

CSI driver for Amazon EBS.

Used in module cloud-provider-aws

Apache License 2.0

Amazon Web Services Node termination handler

Version: v1.25.6

Gracefully handle instance shutdown within Kubernetes for Amazon EC2.

Used in module cloud-provider-aws

Apache License 2.0

Amazon Web Services Terraform provider

Version: 6.69.1

Terraform provider for Amazon Web Services.

Used in module cloud-provider-aws

Mozilla Public License 2.0

cert-manager

Version: 1.21.2

A Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

Used in module cert-manager

Apache License 2.0

cert-manager-webhook-yandex

Version: 1a7dd500a97e9206fa5cfebb102decd3889e3544

ACME DNS-01 webhook for Yandex Cloud DNS.

Used in module cert-manager

Apache License 2.0

Chrony
chrony is a versatile implementation of the Network Time Protocol (NTP)

Used in module chrony

GNU General Public License version 2

Cilium

Version: 1.17.17

Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application workloads such as application containers or processes.

Used in module cni-cilium

Apache License 2.0

Cluster API

Version: 1.12.11

Kubernetes Cluster API for declarative cluster lifecycle management.

Used in module node-manager

Apache License 2.0

Cluster API Provider Metal3

Version: v1.9.5

Infrastructure provider for Kubernetes Cluster API backed by Metal3.

Used in module infra-provider-baremetal

Apache License 2.0

Cluster Autoscaler

Versions: v1.33.1, v1.34.4, v1.35.2, v1.36.1

Kubernetes autoscaler that automatically adjusts the size of the cluster.

Used in module node-manager

Apache License 2.0

CNI plugins

Version: 1.9.1

Some reference and example networking plugins, maintained by the CNI team.

Used in module cni-cilium

Apache License 2.0

CoreDNS
A DNS server/forwarder that chains plugins.

Used in module kube-dns

Apache License 2.0

Descheduler

Version: 0.37.0

A tool to auto-fix scheduling issues Kubernetes does not cover itself.

Used in module descheduler

Apache License 2.0

Dex

Version: 2.46.0

A Federated OpenID Connect Provider with pluggable connectors.

Used in module user-authn

Apache License 2.0

Dynamix Terraform provider

Version: 4.12.2

Terraform provider for Dynamix (DECORT).

Used in module cloud-provider-dynamix

Apache License 2.0

Etcd

Version: 3.6.15

A distributed reliable key-value store for the most critical data of a distributed system.

Used in module control-plane-manager

Apache License 2.0

Flannel

Version: 0.27.4

A simple and easy way to configure a layer 3 network fabric designed for Kubernetes.

Used in module cni-flannel

Apache License 2.0

Gardener Machine Controller Manager

Version: v0.36.0

An operator that manages VMs as a kubernetes custom resource.

Used in module node-manager

Apache License 2.0

Gatekeeper

Version: 3.23.1

Policy-based control for cloud native environments

Used in module admission-policy-engine

Apache License 2.0

Google Cloud Platform Cloud provider

Versions: ccm/v33.4.0, ccm/v34.2.0, ccm/v35.0.5

Cloud Controller Manager for Google Cloud Platform.

Used in module cloud-provider-gcp

Apache License 2.0

Google Cloud Platform CSI driver

Versions: v1.15.4, v1.27.0

CSI driver for Google Compute Engine persistent disks.

Used in module cloud-provider-gcp

Apache License 2.0

Google Cloud Platform Terraform provider

Version: 4.85.0

Terraform provider for Google Cloud Platform.

Used in module cloud-provider-gcp

Mozilla Public License 2.0

Gops

Version: 0.3.27

A tool to list and diagnose Go processes currently running on your system.

Used in module cni-cilium

BSD 3-Clause License

Huawei Cloud provider

Version: v0.26.9

Cloud Controller Manager for Huawei Cloud.

Used in module cloud-provider-huaweicloud

Apache License 2.0

Huawei CSI driver

Version: evs-0.1.11

CSI driver for Huawei Cloud EVS and SFS.

Used in module cloud-provider-huaweicloud

Apache License 2.0

Huawei Terraform provider

Version: 1.69.0

Terraform provider for Huawei Cloud.

Used in module cloud-provider-huaweicloud

Mozilla Public License 2.0

Hubble UI

Version: 0.13.5

Hubble UI is an open-source user interface for Cilium Hubble.

Used in module cilium-hubble

Apache License 2.0

Ironic Python Agent ramdisk

Version: ipa-centos9-stable-2026.1

Default IPA deployment kernel and initramfs.

Used in module infra-provider-baremetal

Apache License 2.0

Ironic Standalone Operator

Version: v0.10.0

Kubernetes operator managing standalone Ironic deployments for Metal3.

Used in module infra-provider-baremetal

Apache License 2.0

Istio

Versions: 1.25.2, 1.27.9, 1.29.6

An open platform to connect, manage, and secure microservices.

Used in module istio

Apache License 2.0

JQ

Version: 1.7.1

A lightweight and flexible command-line JSON processor.

Used in module deckhouse

Mixed (https://github.com/stedolan/jq/blob/master/COPYING)

keepalived

Version: 2.2.7

Provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures.

Used in module keepalived

GNU General Public License v2.0

Kiali

Versions: 2.12.0, 2.21.0, 2.7.1

Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

Used in module istio

Apache License 2.0

Kube-router

Version: 2.0.1

A turnkey solution for Kubernetes networking with aim to provide operational simplicity and high performance.

Used in module network-policy-engine

Apache License 2.0

Kubernetes

Version: 1.33+

An open source system for managing containerized applications across multiple hosts.

Used in module control-plane-manager

Apache License 2.0

Kubernetes Autoscaler

Version: 1.8.1

A autoscaling-related components for Kubernetes.

Used in module vertical-pod-autoscaler

Apache License 2.0

Libcap-ng

Version: 0.8.5

Libcap-ng is a library for Linux that makes using posix capabilities easy..

Used in module openvpn

GNU General Public License v2.0

Metal3 Bare Metal Operator

Version: v0.13.1

Kubernetes operator for provisioning bare metal hosts with Ironic.

Used in module infra-provider-baremetal

Apache License 2.0

Metal3 Ironic Image

Version: v39.0.0

Container image to run OpenStack Ironic as part of Metal3.

Used in module infra-provider-baremetal

Apache License 2.0

MetalLB

Version: 0.15.2

A load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

Used in module metallb

Apache License 2.0

Microsoft Azure Cloud provider

Versions: v1.33.6, v1.34.3, v1.36.1, v1.37.0

Cloud Controller Manager for Microsoft Azure.

Used in module cloud-provider-azure

Apache License 2.0

Microsoft Azure CSI driver

Versions: v1.33.8, v1.34.2, v1.36.0

CSI driver for Microsoft Azure Disks.

Used in module cloud-provider-azure

Apache License 2.0

Microsoft Azure Terraform provider

Version: 3.117.1

Terraform provider for Microsoft Azure Resource Manager.

Used in module cloud-provider-azure

Mozilla Public License 2.0

OAuth2 Proxy

Version: 7.15.5

A reverse proxy and static file server that provides authentication using Providers (Google, GitHub, and others) to validate accounts by email, domain or group.

Used in module user-authn

MIT License

OpenSSL

Version: 3.3.2

TLS/SSL and crypto library.

Used in module openvpn

Apache License 2.0

OpenStack Cloud provider

Versions: v1.33.1, v1.34.1, v1.35.0, v1.36.0

Cloud Controller Manager for OpenStack.

Used in module cloud-provider-openstack

Apache License 2.0

OpenStack Cluster API provider

Version: v0.14.8

Infrastructure provider for Kubernetes Cluster API in OpenStack.

Used in module cloud-provider-openstack

Apache License 2.0

OpenStack CSI driver

Versions: v1.33.1, v1.34.1, v1.35.0, v1.36.0

CSI driver for OpenStack Cinder.

Used in module cloud-provider-openstack

Apache License 2.0

OpenStack Ironic

Version: 752100ac7ff8774510a34963839aaed93bbe8589

Bare metal provisioning service.

Used in module infra-provider-baremetal

Apache License 2.0

OpenStack Ironic Python package

Version: 39.0.0

Python distribution version of the pinned Ironic source commit.

Used in module infra-provider-baremetal

Apache License 2.0

OpenStack Terraform provider

Version: 1.54.1

Terraform provider for OpenStack.

Used in module cloud-provider-openstack

Mozilla Public License 2.0

OpenVPN

Version: 2.6.12

A secure tunneling daemon.

Used in module openvpn

Mixed

pbr

Version: 7.1.3

OpenStack Python build backend used for Ironic.

Used in module infra-provider-baremetal

Apache License 2.0

pip

Version: 25.3

Python package installer used during the Ironic build.

Used in module infra-provider-baremetal

MIT

Python wheels

Version: 0.19

Python dependency wheels supplied by the Deckhouse package manager.

Used in module infra-provider-baremetal

MIT

RPM

Version: rpm-4.20.1-release

Upstream rpm2cpio script used to extract signed EFI boot loaders during the Ironic build.

Used in module infra-provider-baremetal

GPL-2.0-or-later

setuptools

Version: 84.2.1

Python packaging tools used during the Ironic build.

Used in module infra-provider-baremetal

MIT

Shell operator

Version: 1.9.3

A tool for running event-driven scripts in a Kubernetes cluster

Used in module deckhouse

Apache License 2.0

Terraform
A tool for building, changing, and versioning infrastructure safely and efficiently.

Used in module terraform-manager

Mozilla Public License 2.0

VMware Cloud Director Cloud provider

Versions: 1.5.0, 1.6.1

Cloud Controller Manager for VMware Cloud Director.

Used in module cloud-provider-vcd

Apache License 2.0

VMware Cloud Director Cluster API provider

Versions: v1.2.0, v1.3.2

Cluster API provider for VMware Cloud Director.

Used in module cloud-provider-vcd

Apache License 2.0

VMware Cloud Director CSI driver

Versions: 1.4.1, 1.6.0

CSI driver for VMware Cloud Director Named Disks.

Used in module cloud-provider-vcd

Apache License 2.0

VMware Cloud Director Terraform provider

Versions: 3.10.0, 3.14.1

Terraform provider for VMware Cloud Director.

Used in module cloud-provider-vcd

Mozilla Public License 2.0

VMware vSphere Cloud provider

Versions: v1.33.1, v1.34.0, v1.35.1, v1.37.0

Cloud Controller Manager for VMware vSphere.

Used in module cloud-provider-vsphere

Apache License 2.0

VMware vSphere Cluster API provider

Version: v1.15.3

Infrastructure provider for Kubernetes Cluster API in VMware vSphere.

Used in module cloud-provider-vsphere

Apache License 2.0

VMware vSphere CSI driver

Versions: v3.7.1, v3.7.3

CSI driver for VMware vSphere.

Used in module cloud-provider-vsphere

Apache License 2.0

VMware vSphere Terraform provider

Version: 2.0.2

Terraform provider for VMware vSphere.

Used in module cloud-provider-vsphere

Mozilla Public License 2.0

wheel

Version: 0.38.4

Python wheel builder used during the Ironic build.

Used in module infra-provider-baremetal

MIT

Yandex Cloud Cluster API provider

Version: v0.2.0

Infrastructure provider for Kubernetes Cluster API in Yandex Cloud.

Used in module cloud-provider-yandex

Apache License 2.0

Yandex Cloud CSI driver

Version: v0.16.1

CSI driver for Yandex Compute Disks.

Used in module cloud-provider-yandex

Apache License 2.0

Yandex Cloud provider

Version: v0.33.7

Cloud Controller Manager for Yandex Cloud.

Used in module cloud-provider-yandex

Apache License 2.0

Yandex Cloud Terraform provider

Version: 0.174.0

Terraform provider for Yandex Cloud.

Used in module cloud-provider-yandex

Mozilla Public License 2.0

zVirt CSI driver

Version: release-4.22

CSI driver for zVirt (OpenShift oVirt CSI driver).

Used in module cloud-provider-zvirt

Apache License 2.0

zVirt Terraform provider

Version: 2.2.0

Terraform provider for zVirt (oVirt).

Used in module cloud-provider-zvirt

BSD 3-Clause License

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Used directors

If Kubernetes cluster nodes are analyzed by security scanners (antivirus tools), you may need to configure them to avoid false positives.

Deckhouse Platform (DP) uses the following directories when running (download in CSV):

- `/mnt/kubernetes-data` (master node) — only exists in clusters deployed in the cloud when a separate disk is used for the etcd database.
- `/var/lib/etcd` (master node) — etcd database.
- `/var/lib/deckhouse/` (master node) — files of Deckhouse modules, which dynamically loads from a registry.
- `/var/lib/upmeter` (master node, the `upmeter` module) — the upmeter module database.
- `/etc/kubernetes` (any node) — manifests of static pods, PKI certificate files.
- `/var/lib/bashible` (any node) — node configuration files.
- `/var/lib/containerd` (any node) — used to store data related to CRI operation (e.g., containerd); contains container image layers, container file system snapshots, meta information, logs, and other container information.
- `/mnt/vector-data` (any node, the `log-shipper` module) — checkpoints of sent logs.
- `/var/log/containers` (any node) — logs of containers (when using `containerd`).
- `/var/lib/kubelet/` (any node) — `kubelet` configuration files.
- `/opt/cni/bin/` (any node) — CNI plugin executables.
- `/opt/deckhouse/bin/` (any node) — executable files required for Deckhouse to work.
- `/var/log/pods/` (any node) — logs of all pod containers that are running on this cluster node.
- `/etc/cni/` (any node) — CNI plugin configuration files.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

List of reserved Deckhouse Platform DNS names

This page lists the reserved DNS names that Deckhouse Platform (DP) uses. The names are given in the table below.

Use the list if the cluster DNS name template (the `modules.publicDomainTemplate` parameter of the `global` ModuleConfig) is not a wildcard template (for example, `%s-kube.company.my`). In this case, add an A or CNAME record for each name. The record value is the public IP address of the Ingress controller (the HTTP/HTTPS traffic load balancer).

To get the command for adding the DNS names to the local hosts file, use the hosts file data generator.

Hosts file data generator

To access the cluster services without DNS records (for example, from a workstation), add entries for the required names from the table below to the hosts file. Enter your DNS name template (from the `modules.publicDomainTemplate` parameter of the `global` ModuleConfig) and the IP address of the Ingress controller in the generator. The generator returns a ready-to-run command:

DNS name template (publicDomainTemplate) For example, %s.example.c Enter a DNS name template containing `%s`, for example, `%s.domain.my`, `%s-kube.domain.my` or `kube-%s.domain.my`.
IP address of the Ingress controller IP address, for example, 20 Enter a valid IPv4 or IPv6 address.
Generate

List of reserved DNS names

The following table lists the DNS names reserved by DP and the services that use them.

When you add A or CNAME records, use the name from the table in place of `%s` in `publicDomainTemplate`. For example, for the `%s.example.com` template, the `grafana` service gets the name `grafana.example.com`, and for `%s-kube.company.my` — `grafana-kube.company.my`.

Name	Source	Purpose	Note
`api`	The `control-plane-manager` module	Kubernetes API publication. External access to the Kubernetes API via Ingress or Gateway API	To make the name available, enable API publication. The `storage-volume-data-manager` module also uses this name for data export and import (DataExport and DataImport resources). The module obtains the certificate from the Ingress that publishes the Kubernetes API. Therefore, this module also requires API publication
`ai-models`	The `ai-models` module	Model downloads and public model catalog (`/v2`, `/api/distribution/...`)	The public catalog is available when `distribution.mode: PublicCatalog` is set

Name	Source	Purpose	Note
`argocd`	The `operator-argo` module	Argo CD web interface	The name is published only if an Ingress is configured in the ArgoCD resource. The `spec.server.host` parameter sets the name
`code`	Deckhouse Code	Source code management and CI/CD platform	The name is also used to access Code via SSH. You can override the name in the module settings
`code-pages`	Deckhouse Code	GitLab Pages sites	The name is available when Pages are enabled in the CodeInstance resource (`spec.features.pages`). You can override the name with the `spec.features.pages.ingress.hostname` parameter
`code-registry`	Deckhouse Code	Deckhouse Code container image registry	The name is available when the Registry component is enabled in the CodeInstance resource (`spec.features.registry`). You can override the name with the `spec.features.registry.ingress.hostname` parameter
`commander`	The `commander` module	Cluster lifecycle management (creation, updates, inventory)	
`console`	The `console` module	Deckhouse web UI	
`dashboard`	The `dashboard` module	Kubernetes Dashboard web interface for manual cluster management	**Deprecated**. If the `dashboard` module is disabled, the `console` module takes over the `dashboard` name and responds with a `301` redirect to its own address
`ddp`	Deckhouse Development Portal	Deckhouse Development Portal web interface	You can override the name with the `domain.useCustomDomain` and `domain.customDomain` parameters
`dex`	The `user-authn` module	OIDC provider: single sign-on (SSO) to web interfaces and the Kubernetes API. The name is also used by dex-authenticators of other modules and by basic-auth-proxy	

Name	Source	Purpose	Note
`documentation`	The `documentation` module	Web interface with the documentation for the platform version running in the cluster	
`grafana`	The `prometheus` module	Cluster metrics dashboards	If Grafana of the `prometheus` module is not deployed and the `console` module is enabled, the `prometheus` module responds to this name with a redirect to the `console` module
`hubble`	The `cilium-hubble` module	Hubble UI: visualization of network interaction and Cilium network policies	
`istio`	The `istio` module	Kiali web interface: service mesh observability	
`istio-api-proxy`	The `istio` module	Kubernetes API proxy for multicluster: neighboring clusters access the API of this cluster through this proxy	
`istio-metadata`	The `istio` module	Public cluster metadata (root certificate, public keys, API proxy address) for federation and multicluster	
`kubeconfig`	The `user-authn` module	kubeconfig generation web interface for Kubernetes API access	**Deprecated**. Starting from DP version 1.78, the web interface is no longer published
`neuvector`	The `neuvector` module	NeuVector web interface	
`neuvector-api`	The `neuvector` module	NeuVector controller REST API, including the address for connecting other clusters	The name is available when the `controller.apiIngress.enable` parameter is enabled
`observability`	The `observability` module	External read/write access to metrics (`/metrics/main/api/v1/write`)	The name is available when the `externalMetricsAccess.enable` parameter is enabled
`openvpn-admin`	The `openvpn` module	Access to cluster resources via OpenVPN with certificate-based authentication	
`payload-registry`	The `payload-registry` module	Access to payload-registry storage	The name is not published when `https.mode: Disabled`

Name	Source	Purpose	Note
`prometheus`	The `prometheus` module	Prometheus web interface	If Grafana of the `prometheus` module is not deployed and the `console` module is enabled, the `prometheus` module responds to this name with a redirect to the `console` module
`registry`	The `registry` module	External access point to the in-cluster image registry	
`registry-packages-proxy`	The `registry-packages-proxy` module	In-cluster HTTP proxy. Provides access to packages from container image registries	
`status`	The `upmeter` module	Component status page. Shows the availability level over the last 10 minutes	
`stronghold`	Deckhouse Stronghold	Secrets management web interface	
`tools`	The `deckhouse-tools` module	Deckhouse CLI download page	
`upmeter`	The `upmeter` module	Web interface that shows the availability of cluster components	
`virtualization`	DP built-in virtualization	Address for uploading images and disks	

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse CLI overview and installation

Deckhouse CLI (`d8`) is a command-line interface for working with products in the Deckhouse ecosystem. It combines commands for working with Kubernetes, administering Deckhouse Platform, delivering applications, copying images, creating backups, collecting diagnostic information, virtualization, working with user data, and other tasks.

Starting from release 1.59, `d8` is automatically installed on all DP cluster nodes. The utility can also be installed on a separate administrator machine.

Main command groups

`d8` commands are grouped by purpose:

Commands	Purpose
`d8 k`	Commands for managing a cluster (functionally equivalent to `kubectl`).
`d8 system`	Commands for working with platform system settings, configuration, modules, queues, packages, and system logs.
`d8 status`	Command for obtaining the current platform status.
`d8 backup`	Commands for creating backups of key cluster data and generating diagnostic dumps.
`d8 cr`	Commands for working with the container registry.
`d8 mirror`	Commands for copying the platform component images to the local filesystem or a third-party container registry.
`d8 delivery-kit`	Commands for building, publishing, planning, and deploying applications, as well as for working with Helm, registries, SBOM, and related operations.
`d8 data`	Commands for managing data, as well as exporting and importing it.
`d8 v`	Commands for working with virtual machines created a cluster.
`d8 iam`	Commands for managing Deckhouse users, groups, and access grants.
`d8 stronghold`	Commands for managing the lifecycle of secrets (stronghold module)
`d8 network`	Commands for performing network-related operations in the Deckhouse ecosystem, including CNI migration.
`d8 completion`	Commands for generating shell completion scripts.
`d8 tools`	Auxiliary utilities.
`d8 help`	Built-in help.

How do I install the Deckhouse CLI?

There are two ways to install the Deckhouse CLI:

- Starting with version 0.10, you can install using trdl. This method allows you to get fresh versions of the tool with all improvements and fixes.

  > **i** Note that trdl installation requires Internet access to the tuf repository containing the tool. This method will not work in an air-gapped environment!

- You can manually download the executable file and install it on the system.

trdl-based installation

Starting with the Deckhouse CLI version 0.10, you can install it using trdl.

> ⚠ Versions earlier than 0.10 must be uninstalled before proceeding.
>
> If you need to install one of the versions below 0.10, use the outdated installation method.

1. Install trdl client.
2. Add the Deckhouse CLI repository to trdl:

```
URL=https://deckhouse.ru/downloads/deckhouse-cli-trdl
ROOT_VERSION=1
ROOT_SHA512=343bd5f0d8811254e5f0b6fe292372a7b7eda08d276ff255229200f84e58a8151ab2729df3515cb11372dc3
899c70df172a4e54c8a596a73d67ae790466a0491
REPO=d8

trdl add $REPO $URL $ROOT_VERSION $ROOT_SHA512
```

3. Install the latest stable release of the `d8` utility and check if it works as expected:

```
. $(trdl use d8 0 stable) && d8 --version
```

To avoid having to run `. $(trdl use d8 0 stable)` before every Deckhouse CLI invocation, add the following line to your shell's RC file: `alias d8='trdl exec d8 0 stable -- "$@"'`.

Congratulations, you have installed the Deckhouse CLI.

Installing the executable

- Linux x86-64
- macOS x86-64
- macOS ARM64
- Windows

> ⚠ The commands below apply to Deckhouse CLI version v0.34.7, which is bundled with the Deckhouse Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for Linux x86-64 in a convenient way:
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
     3. Click on the button to download Deckhouse CLI for Linux.
   - **From the Deckhouse website**:

```
curl -LO https://deckhouse.io/downloads/deckhouse-cli/v0.34.7/d8-v0.34.7-linux-amd64.tar.gz
```

2. Extract the archive:

```
tar -xvf "d8-v0.34.7-linux-amd64.tar.gz" "linux-amd64/bin/d8"
```

3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

```
sudo mv "linux-amd64/bin/d8" /usr/local/bin/
```

> **i** `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.

4. Check that the CLI is working:

```
d8 help
```

Congrats, you have successfully installed Deckhouse CLI.

> **⚠** The commands below apply to Deckhouse CLI version v0.34.7, which is bundled with the Deckhouse Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for macOS x86-64 in a convenient way:
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
     3. Click on the button to download Deckhouse CLI for macOS.
   - **From the Deckhouse website**:

```
curl -LO https://deckhouse.io/downloads/deckhouse-cli/v0.34.7/d8-v0.34.7-darwin-amd64.tar.gz
```

2. Extract the archive:

```
tar -xvf "d8-v0.34.7-darwin-amd64.tar.gz" "darwin-amd64/bin/d8"
```

3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

```
sudo mv "darwin-amd64/bin/d8" /usr/local/bin/
```

> **i** `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.

4. Check that the CLI is working:

```
d8 help
```

Congrats, you have successfully installed Deckhouse CLI.

⚠ The commands below apply to Deckhouse CLI version v0.34.7, which is bundled with the Deckhouse Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for macOS ARM64 in a convenient way:
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
     3. Click on the button to download Deckhouse CLI for macOS.
   - **From the Deckhouse website**:

     ```
     curl -LO https://deckhouse.io/downloads/deckhouse-cli/v0.34.7/d8-v0.34.7-darwin-arm64.tar.gz
     ```
2. Extract the archive:

   ```
   tar -xvf "d8-v0.34.7-darwin-arm64.tar.gz" "darwin-arm64/bin/d8"
   ```
3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

   ```
   sudo mv "darwin-arm64/bin/d8" /usr/local/bin/
   ```

   i `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.
4. Check that the CLI is working:

   ```
   d8 help
   ```

Congrats, you have successfully installed Deckhouse CLI.

⚠ The commands below apply to Deckhouse CLI version v0.34.7, which is bundled with the Deckhouse Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for Windows x86-64 in a convenient way.
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
     3. Click on the button to download Deckhouse CLI for Windows.
   - **From the Deckhouse website**, download the archive from this link: Windows x86-64.

2. Extract the archive, move the file `d8.exe` to the directory of your choice, and add the directory to the 'PATH' environment variable of the operating system.
3. Unlock the file `d8.exe`, for example, in the following way:
   - Right-click on the file and select *Properties* from the context menu.
   - In the Properties window, make sure you are on the *General* tab.
   - At the bottom of the General tab, you may see a section that says *Security* with a message like *This file came from another computer and might be blocked to help protect this computer.* If this message is displayed, there will be an *Unblock* button or checkbox next to it.
   - Check the *Unblock* checkbox or click the *Unblock* button, then click *Apply* and *OK* to save the changes.
4. Check that the CLI is working:

   ```
   d8 help
   ```

5. Enable auto-completion in PowerShell by running the following command:

   ```
   d8 completion powershell >> $PROFILE
   ```

Congrats, you have successfully installed Deckhouse CLI.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Deckhouse CLI command reference

> This page is generated automatically.

d8 controls the Deckhouse Kubernetes Platform

Version: v0.34.7

Common options:

- `--help`, `-h`
  help for d8
- `--version`, `-v`
  version for d8

d8 backup

Backup various parts of Deckhouse Kubernetes Platform

Show more...

Backup various parts of Deckhouse Kubernetes Platform

© Flant JSC 2025

Options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 backup cluster-config

Usage:

```
d8 backup cluster-config <backup-tarball-path> [options]
```

Take a snapshot of cluster configuration

Show more...

Take a snapshot of cluster configuration.

This command creates a snapshot various kubernetes resources.

© Flant JSC 2025

Options

- `--compress`, `-c`
  Compress result file

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 backup etcd

Usage:

```
d8 backup etcd <snapshot-path> [options]
```

Take a snapshot of ETCD state

Show more...

Take a snapshot of ETCD state.

This command creates a snapshot of the Kubernetes underlying key-value database ETCD.

© Flant JSC 2025

Options

- `--etcd-pod`, `-p`
  Name of the etcd pod to snapshot from. (optional)
- `--verbose`
  Verbose log output.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 backup loki

Usage:

```
d8 backup loki [options]
```

Dump logs from Loki api.

Show more...

Dump Loki logs.

This command dump all logs from Loki api or in given range timestamps in DKP.

© Flant JSC 2025

Options

- `--days`
  Limit maximum number of days in range to output per queue from Loki. (Default 5 maximum number of days in range per queue.)
- `--end`
  Set end timestamp range to dump logs from Loki. (Default get max end timestamp from Loki, ex. "2025-01-14 15:04:05".)
- `--limit`
  Limit the number of lines to output per queue from Loki. (Default 5000 max limit strings per queue.)
- `--start`
  Set start timestamp range to dump logs from Loki. (Default get max start timestamp from Loki, ex. "2025-01-12 15:04:05".)

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 completion

Generate the autocompletion script for the specified shell

Show more...

Generate the autocompletion script for d8 for the specified shell.
See each sub-command's help for details on how to use the generated script.

d8 completion bash

Usage:

```
d8 completion bash [options]
```

Generate the autocompletion script for bash

Show more...

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package.
If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

source <(d8 completion bash)

To load completions for every new session, execute once:

Linux:

d8 completion bash > /etc/bash_completion.d/d8

macOS:

d8 completion bash > $(brew --prefix)/etc/bash_completion.d/d8

You will need to start a new shell for this setup to take effect.

Options

- `--no-descriptions`
  disable completion descriptions

d8 completion fish

Usage:

```
d8 completion fish [options]
```

Generate the autocompletion script for fish

Show more...

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

d8 completion fish | source

To load completions for every new session, execute once:

d8 completion fish > ~/.config/fish/completions/d8.fish

You will need to start a new shell for this setup to take effect.

Options

- `--no-descriptions`
  disable completion descriptions

d8 completion powershell

Usage:

```
d8 completion powershell [options]
```

Generate the autocompletion script for powershell

Show more...

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
d8 completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command
to your powershell profile.

Options

- `--no-descriptions`

  disable completion descriptions

d8 completion zsh

Usage:

```
d8 completion zsh [options]
```

Generate the autocompletion script for zsh

Show more...

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need
to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(d8 completion zsh)
```

To load completions for every new session, execute once:

Linux:

```
d8 completion zsh > "${fpath[1]}/_d8"
```

macOS:

```
d8 completion zsh > $(brew --prefix)/share/zsh/site-functions/_d8
```

You will need to start a new shell for this setup to take effect.

Options

- `--no-descriptions`

  disable completion descriptions

d8 cr

Work with container registries

Show more...

Work with container images in OCI/Docker registries: inspect metadata,
transfer (pull/push), and browse contents.

Authentication uses the Docker config (~/.docker/config.json) - run
"d8 cr login" (or "docker login") first if the registry requires
credentials, or pass --username/--password to any command directly.

Options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)

- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr catalog

Usage:

```
d8 cr catalog REGISTRY [options]
```

List the repositories in a registry

Options

- `--full-ref`

  Print the full repository reference (registry/repo)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr config

Usage:

```
d8 cr config IMAGE [options]
```

Print the config of an image

Show more...

Print the raw config JSON of an image to stdout. Multi-arch indices are
resolved to a single image via --platform.

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr digest

Usage:

```
d8 cr digest [IMAGE] [options]
```

Print the digest of an image

Show more...

By default, fetches the digest of IMAGE from the registry. With --tarball, reads it from a local tarball instead; IMAGE then becomes optional and selects an entry by tag (the first entry is used if omitted).

--full-ref is incompatible with --tarball.

Options

- `--full-ref`

  Print the full image reference with digest (registry/repo@sha256:...); incompatible with --tarball
- `--tarball`

  Read the digest from a local tarball instead of the registry

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr export

Usage:

```
d8 cr export IMAGE [TARBALL] [options]
```

Export the filesystem of an image as a tarball

Show more...

Export writes the merged filesystem of IMAGE as a tar stream to TARBALL (default: "-" = stdout). Output is byte-for-byte equivalent to crane export: linknames are not rewritten, whiteouts are filtered.

For a directory-target extraction with symlink/path-traversal safety checks, use "d8 cr fs extract".

Examples:
  d8 cr export alpine:3.19 -   # stream to stdout
  d8 cr export alpine:3.19 fs.tar   # write to file
  d8 cr export alpine:3.19 - | tar tf -   # list contents

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs

Usage:

```
d8 cr fs [options]
```

Inspect or extract files inside a container image

Show more...

Inspect or extract files inside a container image without running it.

Subcommands show the merged filesystem - what a running container would see,
with deleted files hidden.

Path conventions:
- Input PATH is tolerant: "/etc/nginx", "etc/nginx", and "./etc/nginx"
  are all accepted and refer to the same entry.
- Output paths are tar-relative (no leading "/"), matching the convention
  of "crane export IMAGE - | tar tf -" and POSIX tar archives. JSON output
  uses the same tar-relative form.

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs cat

Usage:

```
d8 cr fs cat IMAGE PATH [options]
```

Print a file from a container image

Show more...

Print a file from a container image to stdout.

Only regular files are supported. Directories, symlinks, hardlinks and
other non-regular entries return an error. Reads the merged filesystem -
files deleted by upper layers are reported as not-found.

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs extract

Usage:

```
d8 cr fs extract IMAGE [options]
```

Extract a container image filesystem to a local directory

Show more...

Extract the merged filesystem of a container image into --output. To
inspect or copy a single file, use "fs cat"; to list a subpath, use
"fs ls IMAGE PATH". An extraction summary is printed when finished.

On interruption (Ctrl+C): partially-written files stay in --output for inspection.
Rerun extract to overwrite them; there is no per-file skip, the full filesystem is
re-materialized from layer 0.

Options

- `--output`, `-o`

  Write the filesystem into this directory

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs ls

Usage:

```
d8 cr fs ls IMAGE [PATH] [options]
```

List files inside a container image

Show more...

List files inside a container image.

PATH limits output to that path and its descendants. Leading "./" or "/" is
stripped, so "/etc" and "etc" are equivalent. Output paths are tar-relative
(no leading "/"), matching "crane export | tar tf -".

Options

- `--long`, `-l`

  Long format with mode and size

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs tree

Usage:

```
d8 cr fs tree IMAGE [PATH] [options]
```

Show the filesystem of a container image as a tree

Show more...

Render the filesystem of a container image as a tree.

PATH (if given) becomes the tree root and is normalized the same way as in
"fs ls" (leading "./" or "/" stripped, "..", trailing "/" cleaned).

The merged filesystem is rendered.

Options

- `--depth`, `-L`

  Max depth to descend (0 = unlimited)
- `--dirsfirst`

  List directories before files
- `--size`

  Show file sizes (human-readable: B / KB / MB)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr login

Usage:

```
d8 cr login [REGISTRY] [options]
```

Log in to a container registry

Show more...

Log in to a container registry.

Verifies the credentials against the registry and, on success, writes them to
the Docker config (~/.docker/config.json, honouring $DOCKER_CONFIG). Every
other "d8 cr" command then authenticates from that config automatically, the
same way "docker login" works.

Credentials come from the --username/--password flags; either may be omitted
to be prompted interactively (the password is read without echo). With no
REGISTRY argument the default Docker Hub registry is used.

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`

  Enable debug logs on stderr

d8 cr ls

Usage:

```
d8 cr ls REPO [options]
```

List the tags in a repository

Options

- `--full-ref`

  Print the full image reference (registry/repo:tag)
- `--omit-digest-tags, -O`

  Skip digest-based tags (sha256-*) created by signing tools

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`

  Enable debug logs on stderr

d8 cr manifest

Usage:

```
d8 cr manifest IMAGE [options]
```

Print the manifest of an image

Show more...

Print the raw manifest bytes of an image to stdout, exactly as the
registry returned them. Suitable for piping to jq or for signature verification.

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`

  Enable debug logs on stderr

d8 cr pull

Usage:

```
d8 cr pull IMAGE... PATH [options]
```

Pull one or more remote images to a local path

Show more...

Pull one or more images and save them to PATH. PATH is a tarball file
for formats "tarball"/"legacy", or a directory for format "oci".

Formats:
  tarball (default) docker-compatible multi-image tarball
  legacy   single-image format compatible with "docker load" (tags only, digests not preserved)
  oci   OCI image-layout directory; keeps all platforms unless --platform is set

When to use:
  tarball - default. Best for shipping one or more images as a single
             file: "docker load" / "podman load" reads it natively.
             Multi-arch indices flatten to one platform - pin it with
             --platform to avoid an ambiguity error.
  oci   - prefer when the destination is OCI-aware tooling (skopeo,
             crane, buildkit, another registry via "cr push --index").
             Preserves the full multi-arch index without --platform.
             Resumable: re-running the same pull skips already-downloaded
             blobs.
  legacy   - last-resort compatibility with very old "docker load" or
             tooling that rejects newer manifests. Lossy: digests are
             not preserved, single-image only. Avoid unless you have a
             specific consumer that fails on tarball.

On interruption (Ctrl+C):
  --format oci   Layer-level resume. Rerun the same pull; existing valid blobs are skipped
                     and in-flight layer temp-files are cleaned up automatically.
  --format tarball| No resume. Rerun replaces the partial archive from scratch.
   legacy
  --cache-path   Layer cache is self-healing: corrupt partial entries are detected on
                     next access and re-downloaded.

Options

- `--cache-path`, `-c`

  Cache image layers under this directory (reused between pulls)
- `--format`

  Output format (one of: tarball, legacy, oci)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr push

Usage:

```
d8 cr push PATH IMAGE [options]
```

Push a local image to a registry

Show more...

A directory PATH is read as an OCI image layout; a file is treated as a
docker-style tarball. Multi-manifest OCI layouts must be pushed with --index.

Prints the pushed reference (with digest) to stdout; use --image-refs to also
write it to a file. --image-refs overwrites the target file if it exists.

Options

- `--image-refs`

  Persist the pushed reference (with digest) to this file
- `--index`

  Push a multi-manifest OCI layout as an index (OCI layout dirs only)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 data

Data operations (export/import)

d8 data create

Usage:

```
d8 data create [flags] data_export_name volume_type/volume_name [options]
```

Deprecated: use 'd8 data export create'

Options

- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live

d8 data delete

Usage:

```
d8 data delete [flags] data_export_name [options]
```

Deprecated: use 'd8 data export delete'

Options

- `--namespace`, `-n`

  data volume namespace

d8 data download

Usage:

```
d8 data download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Deprecated: use 'd8 data export download'

Options

- `--namespace`, `-n`

  data volume namespace
- `--output`, `-o`

  file to save data (default: same as resource)

- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data export

Usage:

```
d8 data export [options]
```

Export data (DataExport)

d8 data export create

Usage:

```
d8 data export create [flags] data_export_name volume_type/volume_name [options]
```

Create dataexport kubernetes resource

Options

- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live

d8 data export delete

Usage:

```
d8 data export delete [flags] data_export_name [options]
```

Delete dataexport kubernetes resource

Options

- `--namespace`, `-n`

  data volume namespace

d8 data export download

Usage:

```
d8 data export download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Download exported data

Options

- `--cleanup`

  Delete auto-created DataExport without prompting (--cleanup=true to delete, --cleanup=false to keep)
- `--namespace`, `-n`

  data volume namespace
- `--output`, `-o`

  file to save data (default: same as resource)
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data export list

Alias: `ls`

Usage:

```
d8 data export list [flags] data_export_name [/path/] [options]
```

List DataExported content information

Options

- `--cleanup`

  Delete auto-created DataExport without prompting (--cleanup=true to delete, --cleanup=false to keep)
- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data import

Usage:

```
d8 data import [options]
```

Import data (DataImport)

d8 data import create

Usage:

```
d8 data import create [flags] data_import_name [options]
```

Create DataImport

Options

- `--file`, `-f`

  PVC manifest file path
- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live
- `--wffc`

  Wait for first consumer

d8 data import delete

Usage:

```
d8 data import delete [flags] data_import_name [options]
```

Delete dataimport kubernetes resource

Options

- `--namespace`, `-n`

  data volume namespace

d8 data import upload

Usage:

```
d8 data import upload [flags] data_import_name path/file.ext [options]
```

Upload a file to the provided url

Options

- `--chunks`, `-c`

  number of chunks to upload

- `--dstPath`, `-d`

  destination path of the uploaded file
- `--file`, `-f`

  file to upload
- `--namespace`, `-n`

  data volume namespace
- `--publish`, `-P`

  publish the uploaded file
- `--resume`

  resume upload if process was interrupted

d8 data list

Usage:

```
d8 data list [flags] data_export_name [/path/] [options]
```

Deprecated: use 'd8 data export list'

Options

- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 delivery-kit

Alias: `dk`

A set of tools for building, distributing, and deploying containerized applications

Show more...

A set of tools for building, distributing, and deploying containerized applications.

LICENSE NOTE: The Deckhouse Delivery Kit functionality is exclusively available to users holding a valid license for any commercial version of the Deckhouse Kubernetes Platform.

© Flant JSC 2026

d8 delivery-kit build

Usage:

```
d8 delivery-kit build [IMAGE_NAME...] [options]
```

Build images

Show more...

Build images that are described in werf.yaml.

The result of build command is built images pushed into the specified repo (or locally if repo is not specified).

If one or more IMAGE_NAME parameters specified, werf will build only these images.

Options

- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--check-built-images`

  Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_CHECK_BUILT_IMAGES)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=...,
  $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa,
  $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit bundle

Usage:

```
d8 delivery-kit bundle [options]
```

Work with werf bundles: publish bundles into container registry and deploy bundles into Kubernetes cluster

d8 delivery-kit bundle apply

Usage:

```
d8 delivery-kit bundle apply [options]
```

Apply bundle into Kubernetes

Show more...

Take latest bundle from the specified container registry using specified version tag or version mask and apply it as a helm chart into Kubernetes cluster.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_ATOMIC by default)
- `--auto-rollback`, `-R`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_AUTO_ROLLBACK by default)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)

- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-create-namespace`

  Don't create the release namespace (default $WERF_NO_CREATE_NAMESPACE)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default $WERF_RELEASE)

- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--render-subchart-notes`

  If set, render subchart notes along with the parent (by default $WERF_RENDER_SUBCHART_NOTES or false)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-deploy-report`

  Save deploy report (by default $WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)

- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--tag`

  Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with

$WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle copy

Usage:

```
d8 delivery-kit bundle copy [options]
```

Copy published bundle into another location

Show more...

Take latest bundle from the specified container registry using specified version tag and copy it
either into a different tag within the same container registry or into another container registry.

Options

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repos
- `--from`

  Source address of the bundle to copy, specify bundle archive using schema `archive:PATH_TO_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.
- `--helm-compatible-chart`, `-C`

  Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370). In helm compatibility mode chart is fully conforming with the helm OCI registry requirements. Default true or $WERF_HELM_COMPATIBLE_CHART.
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--rename-chart`

  Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be set by the $WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--to`

  Destination address of the bundle to copy, specify bundle archive using schema `archive:PATH_TO_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle plan

Usage:

```
d8 delivery-kit bundle plan [options]
```

Plan bundle apply into Kubernetes

Show more...

Take latest bundle from the specified container registry using specified version tag or version mask and plan its deployment as a helm chart into Kubernetes cluster.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--diff-context-lines`

Show N lines of context around diffs ($WERF_DIFF_CONTEXT_LINES by default)

- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--exit-code`

  Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default $WERF_EXIT_CODE or false)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them,

which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver.
  Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)

- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-insignificant-diffs`

  Show insignificant diff lines ($WERF_SHOW_INSIGNIFICANT_DIFFS by default)
- `--show-sensitive-diffs`

  Show sensitive diff lines ($WERF_SHOW_SENSITIVE_DIFFS by default)
- `--show-verbose-crd-diffs`

  Show verbose CRD diff lines ($WERF_SHOW_VERBOSE_CRD_DIFFS by default)
- `--show-verbose-diffs`

  Show verbose diff lines ($WERF_SHOW_VERBOSE_DIFFS by default)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tag`

  Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the

specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle publish

Usage:

```
d8 delivery-kit bundle publish [IMAGE_NAME...] [options]
```

Publish bundle

Show more...

Publish bundle into the container registry. werf bundle contains built images defined in the werf.yaml, Helm chart, Service values which contain built images tags, any custom values and set values params provided during publish invocation, werf addon templates.

Published into container registry bundle can be rolled out by the "werf bundle" command.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)

- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and

development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--helm-compatible-chart`, `-C`

  Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370). In helm compatibility mode chart is fully conforming with the helm OCI registry requirements. Default false or $WERF_HELM_COMPATIBLE_CHART.
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--rename-chart`

  Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be set by the $WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)

- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).

When enabled,
the private signing key must be specified with --sign-key option and
the certificate must be specified with --sign-cert option

- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Publish bundle into container registry repo by the provided tag ($WERF_TAG or latest by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit bundle render

Usage:

```
d8 delivery-kit bundle render [options]
```

Render Kubernetes manifests from bundle

Show more...

Take locally extracted bundle or download bundle from the specified container registry using specified version tag or version mask and render it as Kubernetes manifests.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--bundle-dir`, `-b`

  Get extracted bundle from directory instead of registry (default $WERF_BUNDLE_DIR)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  Include CRDs in the templated output (default $WERF_INCLUDE_CRDS)
- `--insecure-helm-dependencies`

Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)

- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)

- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

  Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--output`

  Write render output to the specified file instead of stdout ($WERF_RENDER_OUTPUT by default)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-only`, `-s`

  only show manifests rendered from the given templates
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tag`

  Provide exact tag version or semver-based pattern, werf will render the latest version of the specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default $WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit chart

Usage:

```
d8 delivery-kit chart [options]
```

Work with charts

d8 delivery-kit chart ts

Usage:

```
d8 delivery-kit chart ts [options]
```

Work with TypeScript charts

d8 delivery-kit chart ts build

Usage:

```
d8 delivery-kit chart ts build [PATH] [options]
```

Build TypeScript chart.

Show more...

Build TypeScript chart in the specified directory. If PATH is not specified, uses the current directory.

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit chart ts init

Usage:

```
d8 delivery-kit chart ts init [PATH] [options]
```

Initialize the files needed to render manifests using TypeScript.

Show more...

Initialize the files needed to render manifests using TypeScript. If PATH is not specified, uses the current directory.

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions

- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`
  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit ci-env

Usage:

```
d8 delivery-kit ci-env CI_SYSTEM [options]
```

Generate werf environment variables for specified CI system

Show more...

Generate werf environment variables for specified CI system.

Currently supported only GitLab (gitlab) and GitHub (github) CI systems

Options

- `--allow-includes-update`
  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--as-env-file`
  Create the .env file and print the path for sourcing (default $WERF_AS_ENV_FILE).
- `--as-file`
  Create the script and print the path for sourcing (default $WERF_AS_FILE).
- `--config`
  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
  Custom path for storing rendered configuration file
- `--config-templates-dir`
  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--debug-templates`
  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`
  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`
  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command will copy specified or default (~/.docker) config to the temporary directory and may perform additional login with new config.
- `--env`
  Use specified environment (default $WERF_ENV)
- `--git-work-tree`
  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--login-to-registry`

  Log in to CI-specific registry automatically if possible (default $WERF_LOGIN_TO_REGISTRY).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to custom file (default $WERF_OUTPUT_FILE_PATH).
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--shell`

  Set to cmdexe, powershell or use the default behavior that is compatible with any unix shell (default $WERF_SHELL).
- `--tagging-strategy`

  stub
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-docker-auth-config`

  Generate Docker config from DOCKER_AUTH_CONFIG environment variable instead of copying current Docker config (default $WERF_USE_DOCKER_AUTH_CONFIG).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit cleanup

Usage:

```
d8 delivery-kit cleanup [options]
```

Cleanup project images in the container registry

Show more...

Safely cleanup unused project images in the container registry.

The command works according to special rules called cleanup policies, which the user defines in werf.yaml.

It is safe to run this command periodically (daily is enough) by automated cleanup job in parallel with other werf commands such as build, converge and host cleanup.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--cleanup-report-path`

  Change cleanup report path (by default $WERF_CLEANUP_REPORT_PATH or ".werf-cleanup-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).

- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and delete images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--keep-list`

  Set path to keep list file (default $WERF_KEEP_LIST)
- `--keep-stages-built-within-last-n-hours`

  Keep stages that were built within last hours (default $WERF_KEEP_STAGES_BUILT_WITHIN_LAST_N_HOURS or 2)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or $WERF_SCAN_CONTEXT_ONLY)
- `--kube-scan-namespaces`

  Kubernetes namespaces to scan for each selected context (can specify multiple). Takes precedence over --scan-context-namespace-only when set.
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)

- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-cleanup-report`

  Save cleanup report (by default $WERF_SAVE_CLEANUP_REPORT or false). Its path configured with --cleanup-report-path
- `--scan-context-namespace-only`

  Scan for used images only in namespace linked with context for each available context in kube-config (or only for the context specified with option --kube-context). When disabled will scan all namespaces in all contexts (or only for the context specified with option --kube-context), unless --kube-scan-namespaces is set. (Default $WERF_SCAN_CONTEXT_NAMESPACE_ONLY)
- `--scan-context-only`

  Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or $WERF_SCAN_CONTEXT_ONLY)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--without-kube`

  Do not skip deployed Kubernetes images (default $WERF_WITHOUT_KUBE)

d8 delivery-kit completion

Usage:

```
d8 delivery-kit completion [options]
```

Generate bash completion scripts

Options

- `--shell`

  Set to bash, zsh, fish or powershell (default $WERF_SHELL or bash)

d8 delivery-kit compose

Usage:

```
d8 delivery-kit compose [options]
```

Work with docker-compose

d8 delivery-kit compose config

Usage:

```
d8 delivery-kit compose config [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-
command-options="OPTIONS"]
```

Run docker-compose config command with forwarded image names.

Show more...

Run docker-compose config command with forwarded image names.
Image environment name format: $WERF_<FORMATTED_WERF_IMAGE_NAME>_DOCKER_IMAGE_NAME
($WERF_DOCKER_IMAGE_NAME for nameless image).
<FORMATTED_WERF_IMAGE_NAME> is werf image name from werf.yaml modified according to the following
rules:
- all characters are uppercase (app -> APP);
- charset /- is replaced with _ (DEV/APP-FRONTEND -> DEV_APP_FRONTEND).
If one or more IMAGE_NAME parameters specified, werf will build and forward only these images.
Given the following werf configuration:
werf.yaml
project: x
configVersion: 1
---
image: frontend
dockerfile: frontend.Dockerfile
---
image: geodata-backend
dockerfile: backend.Dockerfile
Use described images as follows in your docker compose configuration:
docker-compose.yaml
services:
  frontend:
    image: $WERF_FRONTEND_DOCKER_IMAGE_NAME
  backend:
    image: $WERF_GEODATA_BACKEND_DOCKER_IMAGE_NAME

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- $WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose down

Usage:

```
d8 delivery-kit compose down [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"]
```

Run docker-compose down command with forwarded image names.

Show more...

Run docker-compose down command with forwarded image names.
Image environment name format: $WERF_<FORMATTED_WERF_IMAGE_NAME>_DOCKER_IMAGE_NAME ($WERF_DOCKER_IMAGE_NAME for nameless image).
<FORMATTED_WERF_IMAGE_NAME> is werf image name from werf.yaml modified according to the following rules:
- all characters are uppercase (app -> APP);
- charset /- is replaced with _ (DEV/APP-FRONTEND -> DEV_APP_FRONTEND).
If one or more IMAGE_NAME parameters specified, werf will build and forward only these images.
Given the following werf configuration:
werf.yaml
project: x
configVersion: 1
---
image: frontend
dockerfile: frontend.Dockerfile
---
image: geodata-backend
dockerfile: backend.Dockerfile
Use described images as follows in your docker compose configuration:
docker-compose.yaml
services:
  frontend:
    image: $WERF_FRONTEND_DOCKER_IMAGE_NAME
  backend:
    image: $WERF_GEODATA_BACKEND_DOCKER_IMAGE_NAME

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

Enable development mode (default $WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

Container registry storage address (default $WERF_REPO)

- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- $WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose run

Usage:

```
d8 delivery-kit compose run [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] -- SERVICE [COMMAND] [ARGS...]
```

Run docker-compose run command with forwarded image names.

Show more...

Run docker-compose run command with forwarded image names.
Image environment name format: $WERF_<FORMATTED_WERF_IMAGE_NAME>_DOCKER_IMAGE_NAME ($WERF_DOCKER_IMAGE_NAME for nameless image).
<FORMATTED_WERF_IMAGE_NAME> is werf image name from werf.yaml modified according to the following rules:
- all characters are uppercase (app -> APP);
- charset /- is replaced with _ (DEV/APP-FRONTEND -> DEV_APP_FRONTEND).
If one or more IMAGE_NAME parameters specified, werf will build and forward only these images.
Given the following werf configuration:

```
# werf.yaml
project: x
configVersion: 1
---
image: frontend
dockerfile: frontend.Dockerfile
---
image: geodata-backend
dockerfile: backend.Dockerfile
```

Use described images as follows in your docker compose configuration:

```
# docker-compose.yaml
services:
  frontend:
    image: $WERF_FRONTEND_DOCKER_IMAGE_NAME
  backend:
    image: $WERF_GEODATA_BACKEND_DOCKER_IMAGE_NAME
```

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images

until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and

development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or

- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose up

Usage:

```
d8 delivery-kit compose up [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] [--] [SERVICE...]
```

Run docker-compose up command with forwarded image names.

Show more...

Run docker-compose up command with forwarded image names.
Image environment name format: $WERF_<FORMATTED_WERF_IMAGE_NAME>_DOCKER_IMAGE_NAME ($WERF_DOCKER_IMAGE_NAME for nameless image).
<FORMATTED_WERF_IMAGE_NAME> is werf image name from werf.yaml modified according to the following rules:
- all characters are uppercase (app -> APP);
- charset /- is replaced with _ (DEV/APP-FRONTEND -> DEV_APP_FRONTEND).
If one or more IMAGE_NAME parameters specified, werf will build and forward only these images.
Given the following werf configuration:
werf.yaml
project: x
configVersion: 1
---
image: frontend
dockerfile: frontend.Dockerfile
---
image: geodata-backend
dockerfile: backend.Dockerfile
Use described images as follows in your docker compose configuration:
docker-compose.yaml
services:
  frontend:
    image: $WERF_FRONTEND_DOCKER_IMAGE_NAME
  backend:
    image: $WERF_GEODATA_BACKEND_DOCKER_IMAGE_NAME

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or

$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- $WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit config

Usage:

```
d8 delivery-kit config [options]
```

Work with werf.yaml

d8 delivery-kit config graph

Usage:

```
d8 delivery-kit config graph [IMAGE_NAME...] [options]
```

Print dependency graph for images in werf.yaml.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go,

$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit config list

Usage:

```
d8 delivery-kit config list [options]
```

List image and artifact names defined in werf.yaml.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Show only final images
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--images-only`

  Show only final images
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit config render

Usage:

```
d8 delivery-kit config render [IMAGE_NAME...] [options]
```

Render werf.yaml.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit converge

Usage:

```
d8 delivery-kit converge [options]
```

Build and push images, then deploy application into Kubernetes

Show more...

Build and push images, then deploy application into Kubernetes.

The result of converge command is an application deployed into Kubernetes for current git state.
Command will create release and wait until all resources of the release will become ready.

Environment is a required param for the deploy by default, because it is needed to construct Helm
Release name and Kubernetes Namespace. Either --env or $WERF_ENV should be specified for command.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--atomic`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_ATOMIC by default)
- `--auto-rollback`, `-R`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_AUTO_ROLLBACK by default)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=...,

$WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)

- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)

- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-create-namespace`

  Don't create the release namespace (default $WERF_NO_CREATE_NAMESPACE)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)

- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--plan-lifetime`

  How long plan artifact is valid
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver.
  Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--render-subchart-notes`

  If set, render subchart notes along with the parent (by default $WERF_RENDER_SUBCHART_NOTES or false)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)

- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--save-deploy-report`

  Save deploy report (by default $WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.

Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)

- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--use-plan`

  Use the gzip-compressed JSON plan file from the specified path during release install
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit cr

Usage:

```
d8 delivery-kit cr [options]
```

Authenticate with container registry.

d8 delivery-kit cr login

Usage:

```
d8 delivery-kit cr login registry [options]
```

Login into remote registry.

Show more...

Login into remote registry.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--password`, `-p`

  Use specified password for login (default $WERF_PASSWORD)
- `--password-stdin`

  Read password from stdin for login (default $WERF_PASSWORD_STDIN)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--username`, `-u`

  Use specified username for login (default $WERF_USERNAME)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit cr logout

Usage:

```
d8 delivery-kit cr logout registry [options]
```

Logout from a remote registry.

Show more...

Logout from a remote registry.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit dismiss

Usage:

```
d8 delivery-kit dismiss [options]
```

Delete werf release from Kubernetes

Show more...

Delete application from Kubernetes.

Helm Release will be purged and optionally Kubernetes Namespace.

Environment is a required param for the dismiss by default, because it is needed to construct Helm Release name and Kubernetes Namespace. Either --env or $WERF_ENV should be specified for command.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)

- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

Disable explanatory output (default $WERF_QUIET).

- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-deploy-report`

  Use deploy report, previously saved with --save-deploy-report (by default $WERF_USE_DEPLOY_REPORT or

false). Its path and format configured with --deploy-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--with-hooks`

  Delete Helm Release hooks getting from existing revisions (default $WERF_WITH_HOOKS or true)
- `--with-namespace`

  Delete Kubernetes Namespace after purging Helm Release (default $WERF_WITH_NAMESPACE)

d8 delivery-kit export

Usage:

```
d8 delivery-kit export [IMAGE_NAME...] [options]
```

Export images

Show more...

Export images to an arbitrary repository according to a template specified by the --tag option
(build if needed).
The tag may contain the following shortcuts:
- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.
All meta-information related to werf is removed from the exported images, and then images are completely under the user's responsibility.

Options

- `--add-label`

  Add label to exported images (can specify multiple).
  Format: labelName=labelValue[labelName=labelValue ...]. The default separator is a newline ("\n"), but it can be customized using the --add-label-separator flag.
  Also, can be specified with $WERF_EXPORT_ADD_LABEL_* (e.g. $WERF_EXPORT_ADD_LABEL_1=labelName1=labelValue1, $WERF_EXPORT_ADD_LABEL_2=labelName2=labelValue2)
- `--add-label-separator`

  Separator for --add-label values (default $WERF_EXPORT_ADD_LABEL_SEPARATOR or "\n")
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)

- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Set a tag template (can specify multiple).
  It is necessary to use image name shortcut %image% or %image_slug% if multiple images are exported (e.g. REPO:TAG-%image% or REPO-%image%:TAG)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit helm

Usage:

```
d8 delivery-kit helm [options]
```

Manage application deployment with helm

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm create

Usage:

```
d8 delivery-kit helm create NAME [options]
```

create a new chart with the given name

Show more...

This command creates a chart directory along with the common files and
directories used in a chart.

For example, 'helm create foo' will create a directory structure that looks
something like this:

```
foo/
├── .helmignore   # Contains patterns to ignore when packaging Helm charts.
├── Chart.yaml    # Information about your chart
├── values.yaml   # The default values for your templates
├── charts/       # Charts that this chart depends on
└── templates/    # The template files
    └── tests/    # The test files
```

'helm create' takes a path for an argument. If directories in the given path
do not exist, Helm will attempt to create them as it goes. If the given
destination exists and there are files in that directory, conflicting files
will be overwritten, but other files will be left alone.

Options

- `--starter`, `-p`

  the name or absolute path to Helm starter scaffold

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency

Aliases: `dep`, `dependencies`

Usage:

```
d8 delivery-kit helm dependency update|build|list [options]
```

manage a chart's dependencies

Show more...

Manage the dependencies of a chart.

Helm charts store their dependencies in 'charts/'. For chart developers, it is
often easier to manage dependencies in 'Chart.yaml' which declares all

dependencies.

The dependency commands operate on that file, making it easy to synchronize
between the desired dependencies and the actual dependencies stored in the
'charts/' directory.

For example, this Chart.yaml declares two dependencies:

```
# Chart.yaml
dependencies:
- name: nginx
  version: "1.2.3"
  repository: "https://example.com/charts"
- name: memcached
  version: "3.2.1"
  repository: "https://another.example.com/charts"
```

The 'name' should be the name of a chart, where that name must match the name
in that chart's 'Chart.yaml' file.

The 'version' field should contain a semantic version or version range.

The 'repository' URL should point to a Chart Repository. Helm expects that by
appending '/index.yaml' to the URL, it should be able to retrieve the chart
repository's index. Note: 'repository' can be an alias. The alias must start
with 'alias:' or '@'.

Starting from 2.2.0, repository can be defined as the path to the directory of
the dependency charts stored locally. The path should start with a prefix of
"file://". For example,

```
# Chart.yaml
dependencies:
- name: nginx
  version: "1.2.3"
  repository: "file://../dependency_chart/nginx"
```

If the dependency chart is retrieved locally, it is not required to have the
repository added to helm by "helm add repo". Version matching is also supported
for this case.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or
  $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140

- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency build

Usage:

```
d8 delivery-kit helm dependency update|build|list build CHART [options]
```

rebuild the charts/ directory based on the Chart.lock file

Show more...

Build out the charts/ directory from the Chart.lock file.

Build is used to reconstruct a chart's dependencies to the state specified in
the lock file. This will not re-negotiate dependencies, as 'helm dependency update'
does.

If no lock file is found, 'helm dependency build' will mirror the behavior
of 'helm dependency update'.

Options

- `--keyring`

  keyring containing public keys
- `--skip-refresh`

  do not refresh the local repository cache
- `--verify`

  verify the packages against signatures

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency list

Alias: `ls`

Usage:

```
d8 delivery-kit helm dependency update|build|list list CHART [options]
```

list the dependencies for the given chart

Show more...

List all of the dependencies declared in a chart.

This can take chart archives and chart directories as input. It will not alter
the contents of a chart.

This will produce an error if the chart cannot be loaded.

Options

- `--max-col-width`

  maximum column width for output table

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency update

Alias: `up`

Usage:

```
d8 delivery-kit helm dependency update|build|list update CHART [options]
```

update charts/ based on the contents of Chart.yaml

Show more...

Update the on-disk dependencies to mirror Chart.yaml.

This command verifies that the required charts, as expressed in 'Chart.yaml',
are present in 'charts/' and are at an acceptable version. It will pull down
the latest charts that satisfy the dependencies, and clean up old dependencies.

On successful update, this will generate a lock file that can be used to
rebuild the dependencies to an exact version.

Dependencies are not required to be represented in 'Chart.yaml'. For that
reason, an update command will not remove charts unless they are (a) present
in the Chart.yaml file, but (b) at the wrong version.

Options

- `--keyring`

  keyring containing public keys
- `--skip-refresh`

  do not refresh the local repository cache
- `--verify`

  verify the packages against signatures

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm env

Usage:

```
d8 delivery-kit helm env [options]
```

helm client environment information

Show more...

Env prints out all the environment information in use by Helm.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get

Usage:

```
d8 delivery-kit helm get [options]
```

download extended information of a named release

Show more...

This command consists of multiple subcommands which can be used to
get extended information about the release, including:

- The values used to generate the release
- The generated manifest file
- The notes provided by the chart of the release
- The hooks associated with the release
- The metadata of the release

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get all

Usage:

```
d8 delivery-kit helm get all RELEASE_NAME [options]
```

download all information for a named release

Show more...

This command prints a human readable collection of information about the
notes, hooks, supplied values, and generated manifest file of the given release.

Options

- `--revision`

  get the named release with revision
- `--template`

  go template for formatting the output, eg: {{.Release.Name}}

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or

$WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get hooks

Usage:

```
d8 delivery-kit helm get hooks RELEASE_NAME [options]
```

download all hooks for a named release

Show more...

This command downloads hooks for a given release.

Hooks are formatted in YAML and separated by the YAML '---\n' separator.

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or

$WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get manifest

Usage:

```
d8 delivery-kit helm get manifest RELEASE_NAME [options]
```

download the manifest for a named release

Show more...

This command fetches the generated manifest for a given release.

A manifest is a YAML-encoded representation of the Kubernetes resources that were generated from this release's chart(s). If a chart is dependent on other charts, those resources will also be included in the manifest.

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get metadata

Usage:

```
d8 delivery-kit helm get metadata RELEASE_NAME [options]
```

This command fetches metadata for a given release

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  specify release revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get notes

Usage:

```
d8 delivery-kit helm get notes RELEASE_NAME [options]
```

download the notes for a named release

Show more...

This command shows notes provided by the chart of a named release.

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.
Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get values

Usage:

```
d8 delivery-kit helm get values RELEASE_NAME [options]
```

download the values file for a named release

Show more...

This command downloads a values file for a given release.

Options

- `--all`, `-a`

  dump all (computed) values
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get-autogenerated-values

Usage:

```
d8 delivery-kit helm get-autogenerated-values [IMAGE_NAME...] [options]
```

Get service values yaml generated by werf for helm chart during deploy

Show more...

Get service values generated by werf for Helm chart during deploy.

These values includes project name, Docker images IDs and other.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo.

Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--parallel, -p`

Run in parallel (default $WERF_PARALLEL or true)

- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or

- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-namespace

Usage:

```
d8 delivery-kit helm get-namespace [options]
```

Print Kubernetes Namespace that will be used in current configuration with specified params.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

Enable debug (default $WERF_DEBUG).

- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-release

Usage:

```
d8 delivery-kit helm get-release [options]
```

Print Helm Release name that will be used in current configuration with specified params.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm history

Alias: `hist`

Usage:

```
d8 delivery-kit helm history RELEASE_NAME [options]
```

fetch release history

Show more...

History prints historical revisions for a given release.

A default maximum of 256 revisions will be returned. Setting '--max'
configures the maximum length of the revision list returned.

The historical release set is printed as a formatted table, e.g:

```
$ helm history angry-bird
REVISION    UPDATED                     STATUS          CHART           APP VERSION     DESCRIPTION
1           Mon Oct 3 10:15:13 2016     superseded      alpine-0.1.0    1.0             Initial install
2           Mon Oct 3 10:15:13 2016     superseded      alpine-0.1.0    1.0             Upgraded successfully
3           Mon Oct 3 10:15:13 2016     superseded      alpine-0.1.0    1.0             Rolled back to 2
4           Mon Oct 3 10:15:13 2016     deployed        alpine-0.1.0    1.0             Upgraded successfully
```

Options

- `--max`

  maximum number of revision to include in history
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm install

Usage:

```
d8 delivery-kit helm install [NAME] [CHART] [options]
```

install a chart

Show more...

This command installs a chart archive.

The install argument must be a chart reference, a path to a packaged chart,
a path to an unpacked chart directory or a URL.

To override values in a chart, use either the '--values' flag and pass in a file
or use the '--set' flag and pass configuration from the command line, to force
a string value use '--set-string'. You can use '--set-file' to set individual
values from a file when the value itself is too long for the command line
or is dynamically generated. You can also use '--set-json' to set json values
(scalars/objects/arrays) from the command line.

    $ helm install -f myvalues.yaml myredis ./redis

or

    $ helm install --set name=prod myredis ./redis

or

    $ helm install --set-string long_int=1234567890 myredis ./redis

or

    $ helm install --set-file my_script=dothings.sh myredis ./redis

or

    $ helm install --set-json 'master.sidecars=
[{"name":"sidecar","image":"myImage","imagePullPolicy":"Always","ports":
[{"name":"portname","containerPort":1234}]}]' myredis ./redis

You can specify the '--values'/'-f' flag multiple times. The priority will be given to the
last (right-most) file specified. For example, if both myvalues.yaml and override.yaml
contained a key called 'Test', the value set in override.yaml would take precedence:

    $ helm install -f myvalues.yaml -f override.yaml myredis ./redis

You can specify the '--set' flag multiple times. The priority will be given to the
last (right-most) set specified. For example, if both 'bar' and 'newbar' values are
set for a key called 'foo', the 'newbar' value would take precedence:

    $ helm install --set foo=bar --set foo=newbar myredis ./redis

Similarly, in the following example 'foo' is set to '["four"]':

    $ helm install --set-json='foo=["one", "two", "three"]' --set-json='foo=["four"]' myredis ./redis

And in the following example, 'foo' is set to '{"key1":"value1","key2":"bar"}':

    $ helm install --set-json='foo={"key1":"value1","key2":"value2"}' --set-json='foo.key2="bar"' myredis ./redis

To check the generated manifests of a release without installing the chart,
the --debug and --dry-run flags can be combined.

The --dry-run flag will output all generated chart manifests, including Secrets
which can contain sensitive values. Please carefully consider how and when this
flag is used.

If --verify is set, the chart MUST have a provenance file, and the provenance
file MUST pass all verification steps.

There are six different ways you can express the chart you want to install:

1. By chart reference: helm install mymaria example/mariadb
2. By path to a packaged chart: helm install mynginx ./nginx-1.2.3.tgz
3. By path to an unpacked chart directory: helm install mynginx ./nginx
4. By absolute URL: helm install mynginx https://example.com/charts/nginx-1.2.3.tgz
5. By chart reference and repo url: helm install --repo https://example.com/charts/ mynginx nginx
6. By OCI registries: helm install mynginx --version 1.2.3 oci://example.com/charts/nginx

CHART REFERENCES

A chart reference is a convenient way of referencing a chart in a chart repository.

When you use a chart reference with a repo prefix ('example/mariadb'), Helm will look in the local
configuration for a chart repository named 'example', and will then look for a
chart in that repository whose name is 'mariadb'. It will install the latest stable version of that chart
until you specify '--devel' flag to also include development version (alpha, beta, and release candidate releases),
or
supply a version number with the '--version' flag.

To see the list of chart repositories, use 'helm repo list'. To search for
charts in a repository, use 'helm search'.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1,
  $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

  if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this installation when install fails
- `--create-namespace`

  create the release namespace if not present
- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

  add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default
  $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

force resource updates through a replacement strategy
- `--generate-name`, `-g`

  generate the name (and omit the NAME parameter)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be divided by comma.
- `--name-template`

  specify template used to name the release
- `--no-hooks`

  prevent hooks from running during install
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--render-subchart-notes`

  if set, render subchart notes along with the parent
- `--replace`

  re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`

  chart repository url where to locate the requested chart
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`

if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm lint

Usage:

```
d8 delivery-kit helm lint PATH [options]
```

examine a chart for possible issues

Show more...

This command takes a path to a chart and runs a series of tests to verify that
the chart is well-formed.

If the linter encounters things that will cause the chart to fail installation,
it will emit [ERROR] messages. If it encounters issues that break with convention
or recommendation, it will emit [WARNING] messages.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1,
  $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default
  $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--kube-version`

  Kubernetes version used for capabilities and deprecation checks
- `--quiet`

  print only warnings and errors
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with
  $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml,
  $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas:
  key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values
  with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas:
  key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--strict`

  fail on lint warnings
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--with-subcharts`

  lint dependent charts

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm list

Alias: `ls`

Usage:

```
d8 delivery-kit helm list [options]
```

list releases

Show more...

This command lists all of the releases for a specified namespace (uses current namespace context if namespace not specified).

By default, it lists only releases that are deployed or failed. Flags like '--uninstalled' and '--all' will alter this behavior. Such flags can be combined: '--uninstalled --failed'.

By default, items are sorted alphabetically. Use the '-d' flag to sort by release date.

If the --filter flag is provided, it will be treated as a filter. Filters are regular expressions (Perl compatible) that are applied to the list of releases. Only items that match the filter will be returned.

```
$ helm list --filter 'ara[a-z]+'
NAME                UPDATED                                 CHART
maudlin-arachnid    2020-06-18 14:17:46.125134977 +0000 UTC  alpine-0.1.0
```

If no results are found, 'helm list' will exit 0, but with no output (or in the case of no '-q' flag, only headers).

By default, up to 256 items may be returned. To limit this, use the '--max' flag. Setting '--max' to 0 will not return all results. Rather, it will return the server's default, which may be much higher than 256. Pairing the '--max' flag with the '--offset' flag allows you to page through results.

Options

- `--all`, `-a`

  show all releases without any filter applied
- `--all-namespaces`, `-A`

  list releases across all namespaces
- `--date`, `-d`

  sort by release date
- `--deployed`

  show deployed releases. If no other is specified, this will be automatically enabled
- `--failed`

  show failed releases
- `--filter`, `-f`

  a regular expression (Perl compatible). Any releases that match the expression will be included in the results
- `--max`, `-m`

  maximum number of releases to fetch
- `--no-headers`

  don't print headers when using the default output format
- `--offset`

  next release index in the list, used to offset from start value
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pending`

  show pending releases
- `--reverse`, `-r`

  reverse the sort order
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Works only for secret(default) and configmap storage backends.
- `--short`, `-q`

  output short (quiet) listing format
- `--superseded`

  show superseded releases
- `--time-format`

  format time using golang time formatter. Example: --time-format "2006-01-02 15:04:05Z0700"
- `--uninstalled`

  show uninstalled releases (if 'helm uninstall --keep-history' was used)
- `--uninstalling`

show releases that are currently being uninstalled

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm package

Usage:

```
d8 delivery-kit helm package [CHART_PATH] [...] [options]
```

package a chart directory into a chart archive

Show more...

This command packages a chart into a versioned chart archive file. If a path
is given, this will look at that path for a chart (which must contain a
Chart.yaml file) and then package that directory.

Versioned chart archives are used by Helm package repositories.

To sign a chart, use the '--sign' flag. In most cases, you should also

provide '--keyring path/to/secret/keys' and '--key keyname'.

```
$ helm package --sign ./mychart --key mykey --keyring ~/.gnupg/secring.gpg
```

If '--keyring' is not specified, Helm usually defaults to the public keyring
unless your environment is otherwise configured.

Options

- `--app-version`

  set the appVersion on the chart to this version
- `--dependency-update`, `-u`

  update dependencies from "Chart.yaml" to dir "charts/" before packaging
- `--destination`, `-d`

  location to write the chart.
- `--key`

  name of the key to use when signing. Used if --sign is true
- `--keyring`

  location of a public keyring
- `--passphrase-file`

  location of a file which contains the passphrase for the signing key. Use "-" in order to read from stdin.
- `--sign`

  use a PGP private key to sign this package
- `--version`

  set the version on the chart to this semver version

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin

Usage:

```
d8 delivery-kit helm plugin [options]
```

install, list, or uninstall Helm plugins

Show more...

Manage client-side Helm plugins.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin install

Alias: `add`

Usage:

```
d8 delivery-kit helm plugin install [options] <path|url>...
```

install one or more Helm plugins

Show more...

This command allows you to install a plugin from a url to a VCS repo or a local path.

Options

- `--version`
  specify a version constraint. If this is not specified, the latest version is installed

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin list

Alias: `ls`

Usage:

```
d8 delivery-kit helm plugin list [options]
```

list installed Helm plugins

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin uninstall

Aliases: `rm`, `remove`

Usage:

```
d8 delivery-kit helm plugin uninstall <plugin>... [options]
```

uninstall one or more Helm plugins

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin update

Alias: `up`

Usage:

```
d8 delivery-kit helm plugin update <plugin>... [options]
```

update one or more Helm plugins

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm pull

Alias: `fetch`

Usage:

```
d8 delivery-kit helm pull [chart URL | repo/chartname] [...] [options]
```

download a chart from a repository and (optionally) unpack it in local directory

Show more...

Retrieve a package from a package repository, and download it locally.

This is useful for fetching packages to inspect, modify, or repackage. It can
also be used to perform cryptographic verification of a chart without installing
the chart.

There are options for unpacking the chart after download. This will create a
directory for the chart and uncompress into that directory.

If the --verify flag is specified, the requested chart MUST have a provenance
file, and MUST pass the verification process. Failure in any part of this will
result in an error, and the chart will not be saved locally.

Options

- `--ca-file`

verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--destination`, `-d`

  location to write the chart. If this and untardir are specified, untardir is appended to this
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored.
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--prov`

  fetch the provenance file, but don't perform verification
- `--repo`

  chart repository url where to locate the requested chart
- `--untar`

  if set to true, will untar the chart after downloading it
- `--untardir`

  if untar is specified, this flag specifies the name of the directory into which the chart is expanded
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

Set log terminal width.
Defaults to:
* $WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140

- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry

Usage:

```
d8 delivery-kit helm registry [options]
```

login to or logout from a registry

Show more...

This command consists of multiple subcommands to interact with registries.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry login

Usage:

```
d8 delivery-kit helm registry login [host] [options]
```

login to a registry

Show more...

Authenticate to a remote registry.

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify registry client using this SSL certificate file
- `--insecure`

  allow connections to TLS registry without certs
- `--key-file`

  identify registry client using this SSL key file
- `--password`, `-p`

  registry password or identity token
- `--password-stdin`

  read password or identity token from stdin
- `--username`, `-u`

  registry username

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry logout

Usage:

```
d8 delivery-kit helm registry logout [host] [options]
```

logout from a registry

Show more...

Remove credentials stored for a remote registry.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] [options]
```

add, list, remove, update, and index chart repositories

Show more...

This command consists of multiple subcommands to interact with chart repositories.

It can be used to add, remove, list, and index chart repositories.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo add

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] add [NAME] [URL] [options]
```

add a chart repository

Options

- `--allow-deprecated-repos`

  by default, this command will not allow adding official repos that have been permanently deleted. This disables that behavior
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--force-update`

  replace (overwrite) the repo if it already exists
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the repository
- `--key-file`

  identify HTTPS client using this SSL key file
- `--no-update`

  Ignored. Formerly, it would disabled forced updates. It is deprecated by force-update.
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password
- `--password-stdin`

  read chart repository password from stdin
- `--username`

  chart repository username

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo index

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] index [DIR] [options]
```

generate an index file given a directory containing packaged charts

Show more...

Read the current directory and generate an index file based on the charts found.

This tool is used for creating an 'index.yaml' file for a chart repository. To
set an absolute URL to the charts, use '--url' flag.

To merge the generated index with an existing index file, use the '--merge'
flag. In this case, the charts found in the current directory will be merged
into the existing index, with local charts taking priority over existing charts.

Options

- `--json`

  output in JSON format
- `--merge`

  merge the generated index into the given index
- `--url`

  url of chart repository

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo list

Alias: `ls`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] list [options]
```

list chart repositories

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo remove

Alias: `rm`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] remove [REPO1 [REPO2 ...]] [options]
```

remove one or more chart repositories

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo update

Alias: `up`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] update [REPO1 [REPO2 ...]] [options]
```

update information of available charts locally from chart repositories

Show more...

Update gets the latest information about charts from the respective chart repositories.
Information is cached locally, where it is used by commands like 'helm search'.

You can optionally specify a list of repositories you want to update.
 $ helm repo update <repo_name> ...
To update all the repositories, use 'helm repo update'.

Options

- `--fail-on-repo-update-fail`

  update fails if any of the repository updates fail

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm rollback

Usage:

```
d8 delivery-kit helm rollback <RELEASE> [REVISION] [options]
```

roll back a release to a previous revision

Show more...

This command rolls back a release to a previous revision.

The first argument of the rollback command is the name of a release, and the
second is a revision (version) number. If this argument is omitted or set to
0, it will roll back to the previous release.

To see revision numbers, run 'helm history RELEASE'.

Options

- `--cleanup-on-fail`

  allow deletion of new resources created in this rollback when rollback fails
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--dry-run`

  simulate a rollback
- `--force`

  force resource update through delete/recreate if needed
- `--history-max`

  limit the maximum number of revisions saved per release. Use 0 for no limit
- `--no-hooks`

  prevent hooks from running during rollback
- `--recreate-pods`

  performs pods restart for the resource if applicable

- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search

Usage:

```
d8 delivery-kit helm search [keyword] [options]
```

search for a keyword in charts

Show more...

Search provides the ability to search for Helm charts in the various places
they can be stored including the Artifact Hub and repositories you have added.
Use search subcommands to search different locations for charts.

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or
  $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to
  $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search hub

Usage:

```
d8 delivery-kit helm search [keyword] hub [KEYWORD] [options]
```

search for charts in the Artifact Hub or your own hub instance

Show more...

Search for Helm charts in the Artifact Hub or your own hub instance.

Artifact Hub is a web-based application that enables finding, installing, and

publishing packages and configurations for CNCF projects, including publicly available distributed charts Helm charts. It is a Cloud Native Computing Foundation sandbox project. You can browse the hub at https://artifacthub.io/

The [KEYWORD] argument accepts either a keyword string, or quoted string of rich query options. For rich query options documentation, see https://artifacthub.github.io/hub/api/?urls.primaryName=Monocular%20compatible%20search%20API#/Monocular/get_api_chartsvc_v1_charts_search

Previous versions of Helm used an instance of Monocular as the default 'endpoint', so for backwards compatibility Artifact Hub is compatible with the Monocular search API. Similarly, when setting the 'endpoint' flag, the specified endpoint must also be implement a Monocular compatible search API endpoint. Note that when specifying a Monocular instance as the 'endpoint', rich queries are not supported. For API details, see https://github.com/helm/monocular

Options

- `--endpoint`

  Hub instance to query for charts
- `--fail-on-no-result`

  search fails if no results are found
- `--list-repo-url`

  print charts repository URL
- `--max-col-width`

  maximum column width for output table
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search repo

Usage:

```
d8 delivery-kit helm search [keyword] repo [keyword] [options]
```

search repositories for a keyword in charts

Show more...

Search reads through all of the repositories configured on the system, and
looks for matches. Search of these repositories uses the metadata stored on
the system.

It will display the latest stable versions of the charts found. If you
specify the --devel flag, the output will include pre-release versions.
If you want to search using a version constraint, use --version.

Examples:

```
# Search for stable release versions matching the keyword "nginx"
$ helm search repo nginx

# Search for release versions matching the keyword "nginx", including pre-release versions
$ helm search repo nginx --devel

# Search for the latest stable release for nginx-ingress with a major version of 1
$ helm search repo nginx-ingress --version ^1.0.0
```

Repositories are managed with 'helm repo' commands.

Options

- `--devel`

  use development versions (alpha, beta, and release candidate releases), too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--fail-on-no-result`

  search fails if no results are found
- `--max-col-width`

  maximum column width for output table
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--regexp`, `-r`

  use regular expressions for searching repositories you have added
- `--version`

  search using semantic versioning constraints on repositories you have added
- `--versions`, `-l`

  show the long listing, with each version of each chart on its own line, for repositories you have added

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or

$WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret

Usage:

```
d8 delivery-kit helm secret [options]
```

Work with secrets

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret decrypt

Usage:

```
d8 delivery-kit helm secret decrypt [options]
```

Decrypt data

Show more...

Decrypt data from standard input.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret encrypt

Usage:

```
d8 delivery-kit helm secret encrypt [options]
```

Encrypt data

Show more...

Encrypt data from standard input.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and

development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file

Usage:

```
d8 delivery-kit helm secret file [options]
```

Work with secret files

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret file decrypt

Usage:

```
d8 delivery-kit helm secret file decrypt [FILE_PATH] [options]
```

Decrypt secret file data

Show more...

Decrypt data from FILE_PATH or pipe.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file edit

Usage:

```
d8 delivery-kit helm secret file edit FILE_PATH [options]
```

Edit or create new secret file

Show more...

Edit or create new secret file.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file encrypt

Usage:

```
d8 delivery-kit helm secret file encrypt [FILE_PATH] [options]
```

Encrypt file data

Show more...

Encrypt data from FILE_PATH or pipe.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret generate-secret-key

Usage:

```
d8 delivery-kit helm secret generate-secret-key [options]
```

Generate hex encryption key

Show more...

Generate hex encryption key.
For further usage, the encryption key should be saved in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret rotate-secret-key

Usage:

```
d8 delivery-kit helm secret rotate-secret-key [EXTRA_SECRET_VALUES_FILE_PATH...] [options]
```

Regenerate secret files with new secret key

Show more...

Regenerate Secret files with new Secret key.

Old key should be specified in the $WERF_OLD_SECRET_KEY.
New key should reside either in the $WERF_SECRET_KEY or .werf_secret_key file.

Command will extract data with the old key, generate new secret data and rewrite files:
* standard raw Secret files in the .helm/secret folder;
* standard secret Values YAML file .helm/secret-values.yaml;
* additional secret Values YAML files specified with EXTRA_SECRET_VALUES_FILE_PATH params

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values

Usage:

```
d8 delivery-kit helm secret values [options]
```

Work with secret values files

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace, -n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret values decrypt

Usage:

```
d8 delivery-kit helm secret values decrypt [FILE_PATH] [options]
```

Decrypt secret values file data

Show more...

Decrypt data from FILE_PATH or pipe.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values edit

Usage:

```
d8 delivery-kit helm secret values edit FILE_PATH [options]
```

Edit or create new secret values file

Show more...

Edit or create new secret values file.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values encrypt

Usage:

```
d8 delivery-kit helm secret values encrypt [FILE_PATH] [options]
```

Encrypt values file data

Show more...

Encrypt data from FILE_PATH or pipe.
Encryption key should be in $WERF_SECRET_KEY or .werf_secret_key file.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and

development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm show

Alias: `inspect`

Usage:

```
d8 delivery-kit helm show [options]
```

show information of a chart

Show more...

This command consists of multiple subcommands to display information about a chart

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show all

Usage:

```
d8 delivery-kit helm show all [CHART] [options]
```

show all information of the chart

Show more...

This command inspects a chart (directory, file, or URL) and displays all its content
(values.yaml, Chart.yaml, README)

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show chart

Usage:

```
d8 delivery-kit helm show chart [CHART] [options]
```

show the chart's definition

Show more...

This command inspects a chart (directory, file, or URL) and displays the contents
of the Chart.yaml file

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show crds

Usage:

```
d8 delivery-kit helm show crds [CHART] [options]
```

show the chart's CRDs

Show more...

This command inspects a chart (directory, file, or URL) and displays the contents
of the CustomResourceDefinition files

Options

- `--ca-file`

verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show readme

Usage:

```
d8 delivery-kit helm show readme [CHART] [options]
```

show the chart's README

Show more...

This command inspects a chart (directory, file, or URL) and displays the contents
of the README file

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show values

Usage:

```
d8 delivery-kit helm show values [CHART] [options]
```

show the chart's values

Show more...

This command inspects a chart (directory, file, or URL) and displays the contents
of the values.yaml file

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--jsonpath`

  supply a JSONPath expression to filter the output
- `--key-file`

identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm status

Usage:

```
d8 delivery-kit helm status RELEASE_NAME [options]
```

display the status of the named release

Show more...

This command shows the status of a named release.
The status consists of:
- last deployment time
- k8s namespace in which the release lives
- state of the release (can be: unknown, deployed, uninstalled, superseded, failed, uninstalling, pending-install, pending-upgrade or pending-rollback)
- revision of the release
- description of the release (can be completion message or error message, need to enable --show-desc)
- list of resources that this release consists of (need to enable --show-resources)
- details on last test suite run, if applicable
- additional notes provided by the chart

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  if set, display the status of the named release with revision
- `--show-desc`

  if set, display the description message of the named release
- `--show-resources`

  if set, display the resources of the named release

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140

- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm template

Usage:

```
d8 delivery-kit helm template [NAME] [CHART] [options]
```

locally render templates

Show more...

Render chart templates locally and display the output.

Any values that would normally be looked up or retrieved in-cluster will be
faked locally. Additionally, none of the server-side testing of chart validity
(e.g. whether an API is supported) is done.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--api-versions`, `-a`

  Kubernetes api versions used for Capabilities.APIVersions
- `--atomic`

  if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this installation when install fails
- `--create-namespace`

  create the release namespace if not present
- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

  force resource updates through a replacement strategy
- `--generate-name`, `-g`

  generate the name (and omit the NAME parameter)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  include CRDs in the templated output
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--is-upgrade`

  set .Release.IsUpgrade instead of .Release.IsInstall
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--kube-version`

  Kubernetes version used for Capabilities.KubeVersion
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be divided by comma.
- `--name-template`

  specify template used to name the release
- `--no-hooks`

  prevent hooks from running during install
- `--output-dir`

  writes the executed templates to files in output-dir instead of stdout
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--release-name`

  use release name in the output-dir path.
- `--render-subchart-notes`

  if set, render subchart notes along with the parent
- `--replace`

  re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`

chart repository url where to locate the requested chart
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--show-only`, `-s`

  only show manifests rendered from the given templates
- `--skip-crds`

  if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--skip-tests`

  skip tests from templated output
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--validate`

  validate your manifests against the Kubernetes cluster you are currently pointing at. This is the same validation performed on an install
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm test

Usage:

```
d8 delivery-kit helm test [RELEASE] [options]
```

run tests for a release

Show more...

The test command runs the tests for a release.

The argument this command takes is the name of a deployed release.
The tests to be run are defined in the chart that was installed.

Options

- `--filter`

  specify tests by attribute (currently "name") using attribute=value syntax or '!attribute=value' to exclude a test (can specify multiple or separate values with commas: name=test1,name=test2)
- `--logs`

  dump the logs from test pods (this runs after all tests are complete, but before any cleanup)
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm uninstall

Aliases: `del`, `delete`, `un`

Usage:

```
d8 delivery-kit helm uninstall RELEASE_NAME [...] [options]
```

uninstall a release

Show more...

This command takes a release name and uninstalls the release.

It removes all of the resources associated with the last release of the chart
as well as the release history, freeing it up for future use.

Use the '--dry-run' flag to see which releases will be uninstalled without actually
uninstalling them.

Options

- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents. Defaults to background.

- `--description`

  add a custom description
- `--dry-run`

  simulate a uninstall
- `--ignore-not-found`

  Treat "release not found" as a successful uninstall
- `--keep-history`

  remove all associated resources and mark the release as deleted, but retain the release history
- `--no-hooks`

  prevent hooks from running during uninstallation
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--wait`

  if set, will wait until all the resources are deleted before returning. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm upgrade

Usage:

```
d8 delivery-kit helm upgrade [RELEASE] [CHART] [options]
```

upgrade a release

Show more...

This command upgrades a release to a new version of a chart.

The upgrade arguments must be a release and chart. The chart
argument can be either: a chart reference('example/mariadb'), a path to a chart directory,
a packaged chart, or a fully qualified URL. For chart references, the latest
version will be specified unless the '--version' flag is set.

To override values in a chart, use either the '--values' flag and pass in a file
or use the '--set' flag and pass configuration from the command line, to force string
values, use '--set-string'. You can use '--set-file' to set individual
values from a file when the value itself is too long for the command line
or is dynamically generated. You can also use '--set-json' to set json values
(scalars/objects/arrays) from the command line.

You can specify the '--values'/'-f' flag multiple times. The priority will be given to the
last (right-most) file specified. For example, if both myvalues.yaml and override.yaml
contained a key called 'Test', the value set in override.yaml would take precedence:

```
$ helm upgrade -f myvalues.yaml -f override.yaml redis ./redis
```

You can specify the '--set' flag multiple times. The priority will be given to the
last (right-most) set specified. For example, if both 'bar' and 'newbar' values are
set for a key called 'foo', the 'newbar' value would take precedence:

```
$ helm upgrade --set foo=bar --set foo=newbar redis ./redis
```

You can update the values for an existing release with this command as well via the
'--reuse-values' flag. The 'RELEASE' and 'CHART' arguments should be set to the original
parameters, and existing values will be merged with any values set via '--values'/'-f'
or '--set' flags. Priority is given to new values.

```
$ helm upgrade --reuse-values --set foo=bar --set foo=newbar redis ./redis
```

The --dry-run flag will output all generated chart manifests, including Secrets
which can contain sensitive values. Please carefully consider how and when this
flag is used.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1,
  $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

  if set, upgrade process rolls back changes made in case of failed upgrade. The --wait flag will be set
  automatically if --atomic is used
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this upgrade when upgrade fails
- `--create-namespace`

  if --install is set, create the release namespace if not present

- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

  add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the upgrade process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

  force resource updates through a replacement strategy
- `--history-max`

  limit the maximum number of revisions saved per release. Use 0 for no limit
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--install`, `-i`

  if a release by this name doesn't already exist, run an install
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be separated by comma. Original release labels will be merged with upgrade labels. You can unset label using null.
- `--no-hooks`

  disable pre/post upgrade hooks
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--recreate-pods`

  performs pods restart for the resource if applicable
- `--render-subchart-notes`

  if set, render subchart notes along with the parent
- `--repo`

  chart repository url where to locate the requested chart
- `--reset-then-reuse-values`

  when upgrading, reset the values to the ones built into the chart, apply the last release's values and merge

in any overrides from the command line via --set and -f. If '--reset-values' or '--reuse-values' is specified, this is ignored
- `--reset-values`

  when upgrading, reset the values to the ones built into the chart
- `--reuse-values`

  when upgrading, reuse the last release's values and merge in any overrides from the command line via --set and -f. If '--reset-values' is specified, this is ignored
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`

  if set, no CRDs will be installed when an upgrade is performed with install flag enabled. By default, CRDs are installed if not already present, when an upgrade is performed with install flag enabled
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm verify

Usage:

```
d8 delivery-kit helm verify PATH [options]
```

verify that a chart at the given path has been signed and is valid

Show more...

Verify that the given chart has a valid provenance file.

Provenance files provide cryptographic verification that a chart has not been
tampered with, and was packaged by a trusted provider.

This command can be used to verify a local chart. Several other commands provide
'--verify' flags that run the same validation. To generate a signed package, use
the 'helm package --sign' command.

Options

- `--keyring`

  keyring containing public keys

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm version

Usage:

```
d8 delivery-kit helm version [options]
```

print the client version information

Show more...

Show the version for Helm.

This will print a representation the version of Helm.
The output will look something like this:

version.BuildInfo{Version:"v3.2.1", GitCommit:"fe51cd1e31e6a202cba7dead9552a6d418ded79a",
GitTreeState:"clean", GoVersion:"go1.13.10"}

- Version is the semantic version of the release.
- GitCommit is the SHA for the commit that this version was built from.
- GitTreeState is "clean" if there are no local code changes when this binary was
  built, and "dirty" if the binary was built from locally modified code.
- GoVersion is the version of Go that was used to compile Helm.

When using the --template flag the following properties are available to use in
the template:

- .Version contains the semantic version of Helm
- .GitCommit is the git commit
- .GitTreeState is the state of the git tree when Helm was built
- .GoVersion contains the version of Go that Helm was compiled with

For example, --template='Version: {{.Version}}' outputs 'Version: v3.2.1'.

Options

- `--client`, `-c`

  display client version information
- `--short`

  print the version number
- `--template`

  template for version string format

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit host

Usage:

```
d8 delivery-kit host [options]
```

Work with werf cache and data of all projects on the host machine

d8 delivery-kit host cleanup

Usage:

```
d8 delivery-kit host cleanup [options]
```

Cleanup old unused werf cache and data of all projects on host machine.

Show more...

Cleanup old unused werf cache and data of all projects on host machine.

The data include:
* Lost Docker containers and images from interrupted builds.
* Old service tmp dirs, which werf creates during every build, converge and other commands.
* Local cache:
  * remote Git clones cache;
  * Git worktree cache.

It is safe to run this command periodically by automated cleanup job in parallel with other werf commands such as build, converge and cleanup.

Options

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--force`

  Force deletion of images which are being used by some containers (default $WERF_FORCE)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit host purge

Usage:

```
d8 delivery-kit host purge [options]
```

Purge werf images, cache and other data for all projects on host machine

Show more...

Purge werf images, cache and other data for all projects on host machine.

The data include:
* Old service tmp dirs, which werf creates during every build, converge and other commands.
* Local cache:
  * remote Git clones cache;
  * Git worktree cache.
* Shared context:
  * Mounts which persists between several builds (mounts from build_dir).

WARNING: Do not run this command during any other werf command is working on the host machine. This command is supposed to be run manually.

Options

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--force`

  First remove containers that use werf docker images which are going to be deleted
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes

Usage:

```
d8 delivery-kit includes [options]
```

Work with werf includes

d8 delivery-kit includes get-file

Usage:

```
d8 delivery-kit includes get-file [FILE_NAME] [options]
```

Display file content that will be used by werf according to the includes overlay rules.

Show more...

Display file content that will be used by werf according to the includes overlay rules.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

Enable debug (default $WERF_DEBUG).

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

Disable explanatory output (default $WERF_QUIET).

- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes ls-files

Usage:

```
d8 delivery-kit includes ls-files [GLOB...] [options]
```

List files in the project according to the includes overlay rules.

Show more...

List files in the project according to the includes overlay rules.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--filter`

  Filter by source, e.g. --filter=source=local,remoteRepo (default $WERF_INCLUDES_LIST_FILTER or all sources).
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default

$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--ssh-key`

  Use only specific ssh key(s).

  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes update

Usage:

```
d8 delivery-kit includes update [options]
```

Create or update includes lock file (default: werf-includes.lock).

Show more...

Create or update includes lock file by resolving git references in the includes config to their latest commits (default: werf-includes.lock).

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit kube-run

Usage:

```
d8 delivery-kit kube-run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image in Kubernetes

Show more...

Run container in Kubernetes for specified project image from werf.yaml (build if needed).

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--auto-pull-secret`

  Automatically create docker config secret in the namespace and plug it via pod's imagePullSecrets for private registry access (default $WERF_AUTO_PULL_SECRET or true if not specified)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--copy-from`

  Copy file/dir from container to local machine after user command execution. Example: "/from/file:to". Can be specified multiple times. Can also be defined with "$WERF_COPY_FROM_*", e.g. "WERF_COPY_FROM_1=from:to".
- `--copy-to`

  Copy file/dir from local machine to container before user command execution. Example: "from:/to/file". Can be specified multiple times. Can also be defined with "$WERF_COPY_TO_*", e.g. "WERF_COPY_TO_1=from:to".
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-options`

  Pass extra options to "kubectl run" command, which will create a Pod (default $WERF_EXTRA_OPTIONS)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--interactive`, `-i`

  Enable interactive mode (default $WERF_INTERACTIVE or false if not specified)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--overrides`

  Inline JSON to override/extend any fields in created Pod, e.g. to add imagePullSecrets field (default $WERF_OVERRIDES). %pod_name%, %container_name%, and %container_image% will be replaced with the names of the created pod, container, and container image, respectively.
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--pod`

  Set created pod name (default $WERF_POD or autogenerated if not specified)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--rm`

  Remove pod and other created resources after command completion (default $WERF_RM or true if not specified)
- `--rm-with-namespace`

  Remove also a namespace after command completion (default $WERF_RM_WITH_NAMESPACE or false if not specified)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option

- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--tty`, `-t`

  Allocate a TTY (default $WERF_TTY or false if not specified)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false).
  Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit lint

Usage:

```
d8 delivery-kit lint [IMAGE_NAME...] [options]
```

Lint Helm chart

Show more...

Lint Helm chart. This command will calculate digests and build (if needed) all images defined in the werf.yaml.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)

- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)

- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

  Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--lookup-resources`

  Manifest files used as a cluster stub for the lookup template function in non-remote mode. Multi-document and kind:List supported. Namespaced resources must set metadata.namespace, otherwise namespace-scoped lookups ignore the requested namespace. Can be also set with $WERF_LOOKUP_RESOURCES_* environment variables (can specify multiple)
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--parallel`, `-p`

Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)

- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

Use stubs instead of real tags (default $WERF_STUB_TAGS)

- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default $WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit managed-images

Usage:

```
d8 delivery-kit managed-images [options]
```

Work with managed images which will be preserved during cleanup procedure

d8 delivery-kit managed-images add

Usage:

```
d8 delivery-kit managed-images add [options]
```

Add image record to the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and write images to the specified repo
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit managed-images ls

Usage:

```
d8 delivery-kit managed-images ls [options]
```

List managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file

- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read images from the specified repo
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)

- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)

- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit managed-images rm

Usage:

```
d8 delivery-kit managed-images rm [options]
```

Remove image record from the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and write images to the specified repo
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit plan

Usage:

```
d8 delivery-kit plan [options]
```

Prepare deploy plan and show how resources in a Kubernetes cluster would change on next deploy

Show more...

Prepare deploy plan and show how resources in a Kubernetes cluster would change on next deploy.

Environment is a required param by default, because it is needed to construct Helm Release name and Kubernetes Namespace. Either --env or $WERF_ENV should be specified for command.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")

- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or

$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--diff-context-lines`

  Show N lines of context around diffs ($WERF_DIFF_CONTEXT_LINES by default)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--exit-code`

  Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default $WERF_EXIT_CODE or false)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)

- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used

to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver.
  Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g.
  $WERF_RELEASE_LABEL_1=labelName1=labelValue1,
  $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--save-plan`

  Save the gzip-compressed JSON install plan to the specified file
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=...,
  $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)

- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-insignificant-diffs`

  Show insignificant diff lines ($WERF_SHOW_INSIGNIFICANT_DIFFS by default)
- `--show-plan`

  Show plan artifact planned changes
- `--show-sensitive-diffs`

  Show sensitive diff lines ($WERF_SHOW_SENSITIVE_DIFFS by default)
- `--show-verbose-crd-diffs`

  Show verbose CRD diff lines ($WERF_SHOW_VERBOSE_CRD_DIFFS by default)
- `--show-verbose-diffs`

  Show verbose diff lines ($WERF_SHOW_VERBOSE_DIFFS by default)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)

- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  No-op
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  No-op
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit purge

Usage:

```
d8 delivery-kit purge [options]
```

Purge all project images in the container registry

Show more...

Purge all project images in the container registry.

WARNING: Images that are being used in the Kubernetes cluster will also be deleted.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo.

Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)

- `--cleanup-report-path`

  Change cleanup report path (by default $WERF_CLEANUP_REPORT_PATH or ".werf-cleanup-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to delete images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)

- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-cleanup-report`

  Save cleanup report (by default $WERF_SAVE_CLEANUP_REPORT or false). Its path configured with --cleanup-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit render

Usage:

```
d8 delivery-kit render [IMAGE_NAME...] [options]
```

Render Kubernetes templates

Show more...

Render Kubernetes templates. This command will calculate digests and build (if needed) all images defined in the werf.yaml.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)

- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  Include CRDs in the templated output (default $WERF_INCLUDE_CRDS)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

  Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--lookup-resources`

  Manifest files used as a cluster stub for the lookup template function in non-remote mode. Multi-document and kind:List supported. Namespaced resources must set metadata.namespace, otherwise namespace-scoped lookups ignore the requested namespace. Can be also set with $WERF_LOOKUP_RESOURCES_* environment variables (can specify multiple)
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--output`

  Write render output to the specified file instead of stdout ($WERF_RENDER_OUTPUT by default)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)

- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.global.werf.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-only`, `-s`

only show manifests rendered from the given templates

- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  No-op
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:

- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.
Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")

- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default $WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit rollback

Usage:

```
d8 delivery-kit rollback [options]
```

Rollback the Helm release

Show more...

Rollback the Helm release.

Rollback to the previous revision by default. Use --revision to specify the revision to rollback to. Should be run inside of a werf project directory, otherwise --namespace and --release must be specified.

The result of rollback command is an application deployed into Kubernetes. Command will create the new release based on the one we rollback to and wait until all resources of the release will become ready.

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).

- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)

- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--revision`

  Revision number to rollback to (if not specified, rolls back to previous revision)
- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--rollback-report-path`

  Change rollback report path and format (by default $WERF_ROLLBACK_REPORT_PATH or ".werf-rollback-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-rollback-report`

  Save rollback report (by default $WERF_SAVE_ROLLBACK_REPORT or false). Its path and format configured with --rollback-report-path
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit run

Usage:

```
d8 delivery-kit run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image

Show more...

Run container for specified project image from werf.yaml (build if needed).

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bash`

  Use predefined docker options and command for debug
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read and pull images from the specified repo

- `--docker-options`
  Define docker run options (default $WERF_DOCKER_OPTIONS)
- `--dry-run`
  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`
  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`
  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`
  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
  Use specified environment (default $WERF_ENV)
- `--final-repo`
  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`
  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`
  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`
  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
  No-op
- `--insecure-registry`
  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or

$WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)

- `--shell`

  Use predefined docker options and command for debug
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom

Usage:

```
d8 delivery-kit sbom [options]
```

Work with SBOM artifacts

d8 delivery-kit sbom get

Usage:

```
d8 delivery-kit sbom get [IMAGE_NAME] [options]
```

Get SBOM of an image

Show more...

Get SBOM of an image that is described in werf.yaml and prints the result to stdout.

SBOM is stored as an OCI artifact attached to the container image in the container registry, so --repo is required.
If SBOM is not found, the command triggers werf build to generate one.

A specific version of the image may be selected with --tag or --digest; these flags are mutually exclusive.

Options

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=...,

$WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--digest`

  Digest of the image to get SBOM for (mutually exclusive with --tag)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)

- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

Set a specific project name (default $WERF_PROJECT_NAME)

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Content-based tag of the image to get SBOM for (mutually exclusive with --digest)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom merge

Usage:

```
d8 delivery-kit sbom merge [options]
```

Merge per-image SBOMs into a product-level SBOM

Show more...

Merge per-image CycloneDX 1.6 SBOMs into a single product/module-level SBOM.

Takes a JSON mapping file (image name -> sha256 digest) as input, pulls per-image SBOMs from the container registry, and produces an aggregated SBOM with preserved dependency graphs.

Two ISPRAS-defined output formats are supported:
- "container": hierarchical — each image becomes a top-level container component with nested packages.
- "oss": flat — all packages from all images are merged into a deduplicated flat list.

GOST properties (attack_surface, security_function) are aggregated bottom-up using the "yes > indirect > no" precedence rule.

The flags --input, --ispras-format, --app-name, --app-version and --manufacturer are required. The merged SBOM is written to stdout unless --output is given.

Options

- `--app-name`

  Application/product name for the merged SBOM metadata
- `--app-version`

  Application/product version for the merged SBOM metadata
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)

  Command needs granted permissions to pull SBOM artifacts from the specified repo
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--input`

  Path to JSON mapping file (image name -> sha256:digest)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--ispras-format`

  ISPRAS SBOM format: "oss" or "container"
- `--log-color-mode`

  Set log color mode.

  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--manufacturer`

  Manufacturer name for the merged SBOM metadata
- `--output`, `-o`

  Output file path (defaults to stdout)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.

  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit sbom validate

Usage:

```
d8 delivery-kit sbom validate [options]
```

Validate SBOM file against ISPRAS schemas

Show more...

Validate CycloneDX JSON SBOM files against ISPRAS schemas using sbom-checker.

The command runs sbom-checker inside a Docker container and reports validation results. Supports both OSS and container SBOM types.

The flags --path and --ispras-format are required. Repeat --path to validate several files in one run. Pass --check-vcs to additionally validate VCS URLs.

Options

- `--check-vcs`

  Enable VCS URL validation
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to pull the ISPRAS SBOM checker image
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--ispras-format`

  ISPRAS SBOM format: oss or container
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--path`

  Path to CycloneDX JSON SBOM file (repeatable)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit slugify

Usage:

```
d8 delivery-kit slugify STRING [options]
```

Print slugged string by specified format.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).

- `--format`, `-f`

  r|helm-release: suitable for Helm Release
  ns|kubernetes-namespace: suitable for Kubernetes Namespace
  tag|docker-tag: suitable for Docker Tag
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit stages

Usage:

```
d8 delivery-kit stages [options]
```

Work with werf stages

d8 delivery-kit stages copy

Usage:

```
d8 delivery-kit stages copy [options]
```

Copy stages between container registry and archive

Show more...

Copy project stages between storages.

Supported copy scenarios:
• Container registry to container registry
• Container registry to archive
• Archive to container registry

By default copies all project stages. Use --all=false to copy only stages relevant for the current git commit. This requires a working directory with git repository and performs build to identify current stages.

Note: this flag is ignored when copying from archive to container registry.

Options

- `--all`

  Copy all project stages (default: true). Use --all=false to copy stages for current build only. Note: this flag is ignored when copying from archive to container registry.
- `--allow-includes-update`

Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)

- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--build-report-operations`

  Collect low-level operations statistics: add Operations and StageCache sections to the build report and print operations summary after the build (by default $WERF_BUILD_REPORT_OPERATIONS or false). Also enabled by --log-debug
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo. Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repos
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

Container registry storage address (default $WERF_FINAL_REPO)

- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--from`

  Source address to copy stages from. Use archive:PATH for stage archive or [docker://]REPO for container registry.
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)

- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--to`

  Destination address to copy stages to. Use archive:PATH for stage archive or [docker://]REPO for container registry.
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit synchronization

Usage:

```
d8 delivery-kit synchronization [options]
```

Run synchronization server

Show more...

Run synchronization server

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--host`

  Bind synchronization server to the specified host (default localhost or $WERF_HOST)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kubernetes`

Use kubernetes lock-manager stages-storage-cache (default $WERF_KUBERNETES)

- `--kubernetes-namespace-prefix`

  Use specified prefix for namespaces created for lock-manager and stages-storage-cache (defaults to 'werf-synchronization-' when --kubernetes option is used or $WERF_KUBERNETES_NAMESPACE_PREFIX)
- `--local`

  Use file lock-manager and file stages-storage-cache (true by default or $WERF_LOCAL)
- `--local-lock-manager-base-dir`

  Use specified directory as base for file lock-manager (~/.werf/synchronization_server/lock_manager by default or $WERF_LOCAL_LOCK_MANAGER_BASE_DIR)
- `--local-stages-storage-cache-base-dir`

  Use specified directory as base for file stages-storage-cache (~/.werf/synchronization_server/stages_storage_cache by default or $WERF_LOCAL_STAGES_STORAGE_CACHE_BASE_DIR)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--port`

  Bind synchronization server to the specified port (default 55581 or $WERF_PORT)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--ttl`

  Time to live for lock-manager locks and stages-storage-cache records (default $WERF_TTL)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit verify

Usage:

```
d8 delivery-kit verify --image-ref <image-ref> [--image-ref <image-ref>...] --verify-roots=<root-ca.pem> [--verify-manifest] [--verify-elf-files] [options]
```

Verify images

Show more...

Verify the integrity of built images by checking signatures of manifests and ELF files using specific image references.

The command outputs detailed logs about the verification process.

Options

- `--container-registry-mirror`
  (Buildah-only) Use specified mirrors for docker.io
- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--docker-config`
  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--home-dir`
  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--image-ref`
  Verify only passed references (default $WERF_IMAGE_REF separated by comma)
- `--insecure-registry`
  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`
  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--parallel, -p`
  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`
  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`
  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`
  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).
- `--verify-bsign-elf-files`
  Enable ELF files verification for bsign signed files (default $WERF_VERIFY_BSIGN_ELF_FILES).
- `--verify-elf-files`
  Enable ELF files verification (default $WERF_VERIFY_ELF_FILES).
  When enabled, the root certificates must be specified with --verify-roots option
- `--verify-manifest`
  Enable image manifest verification (default $WERF_VERIFY_MANIFEST).

When enabled,
the root certificates must be specified with --verify-roots option

- `--verify-roots`

  The root certificates as path to PEM file or base64-encoded PEM (default $WERF_VERIFY_ROOTS separated by comma)

d8 delivery-kit version

Usage:

```
d8 delivery-kit version [options]
```

Print version

d8 dist

Manage the d8 distribution: the deckhouse-cli binary and its plugins

Show more...

Manage the d8 distribution - the deckhouse-cli binary and its plugins.

See the state of the distribution with 'd8 dist status': the d8 version, installed
plugins, and what is outdated.

Versions are served by the in-cluster registry-packages-proxy, authenticated by the
current kubeconfig identity.

Update the binary with 'd8 dist update'; manage plugins under 'd8 dist plugins'.

Environment variables:
D8_RPP_ENDPOINT registry-packages-proxy base URL (otherwise discovered from the cluster)
D8_RPP_CA_FILE PEM CA bundle to verify the proxy TLS certificate
KUBECONFIG path to the kubeconfig file

Options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist check

Usage:

```
d8 dist check [options]
```

Report whether a newer deckhouse-cli version is available

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.

- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist plugins

Usage:

```
d8 dist plugins [options]
```

Manage Deckhouse CLI plugins

Show more...

Manage Deckhouse CLI plugins.

Plugins are pulled from the in-cluster registry-packages-proxy, authenticated by the
current kubeconfig identity.

Installing a plugin that is already present updates it: 'd8 dist plugins install <name>',
or 'd8 dist plugins install --all' for every installed plugin at once.

Environment variables:
  D8_PLUGINS_SKIP_CLUSTER_CHECKS=1 skip cluster-side plugin requirement checks
  DECKHOUSE_CLI_PATH   plugins directory (same as --plugins-dir)
  D8_RPP_ENDPOINT   registry-packages-proxy base URL
  D8_RPP_CA_FILE   PEM CA bundle for proxy TLS verification
  KUBECONFIG   path to the kubeconfig file

Options

- `--insecure`

  Interact with the source registry over HTTP.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.

- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist plugins contract

Usage:

```
d8 dist plugins contract <plugin-name> [options]
```

Show a plugin's contract

Show more...

Show the latest published contract of a plugin: version, description and requirements.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist plugins install

Usage:

```
d8 dist plugins install <plugin-name> | --all [options]
```

Install or update Deckhouse CLI plugins

Show more...

Install a plugin: the newest version compatible with this cluster by default,

an exact one with --version.

Installing a plugin that is already present updates it, so this is also how a
plugin is brought up to date; --all does that for every installed plugin at once,
each within its own current major.

Plugins this one depends on are installed/upgraded automatically. With --use-major
dependencies may also cross their own major to satisfy a constraint.

A version already on disk is activated by repointing the 'current' symlink -
no download. Plugin requirements are always checked before the switch.

Options

- `--all`

  Update every installed plugin instead of naming one, each within its own current major.
- `--force`

  Reinstall even if the selected version is already installed (re-pull + re-verify).
- `--use-major`

  Pin to a specific major version. By default an install stays within the installed plugin's major; pass this to cross majors (dependencies may cross theirs too).
- `--version`

  Exact version to install. Skips compatibility selection and may install a pre-release.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist plugins list

Usage:

```
d8 dist plugins list [options]
```

List Deckhouse CLI plugins

Show more...

Show the plugins installed locally and, when the transport can enumerate them, the plugins published in the registry and ready to install.

The published set is read from the tags of the plugins repository, where each plugin has an image tagged with its name. Only direct registry access (--source) reaches that repository: the registry-packages-proxy addresses plugins by exact name and does not serve the plugins path, so over the proxy this half is reported as unsupported rather than attempted.

Whatever cannot be resolved is reported in place rather than dropped: a plugin whose versions cannot be read is still listed by name, and an unreachable registry leaves the installed list intact.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist plugins remove

Aliases: `uninstall`, `delete`

Usage:

```
d8 dist plugins remove <plugin-name> [options]
```

Remove an installed plugin

Show more...

Remove an installed plugin from disk.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist plugins remove all

Usage:

```
d8 dist plugins remove <plugin-name> all [options]
```

Remove all installed plugins

Show more...

Remove all plugins from the Deckhouse CLI at once

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.

- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist plugins versions

Usage:

```
d8 dist plugins versions <plugin-name> [options]
```

List all versions of a plugin

Show more...

List all published versions of a plugin, newest first. The installed version is marked, versions newer than it are highlighted.

A plugin is published one image per platform, so a release reaches the registry as several tags. They are collapsed into one line per version, listing the platforms that version was built for.

Versions are fetched by the plugin's name through the registry-packages-proxy, so no catalog access is needed. Install a specific version with 'd8 dist plugins install <name> --version X' - a version already on disk is switched to instantly, without a download.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure`

  Interact with the source registry over HTTP.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--license`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=. Defaults to $D8_MIRROR_LICENSE_TOKEN.
- `--plugins-dir`

  Path to the d8 plugins directory. Defaults to $DECKHOUSE_CLI_PATH.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.

- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--skip-cluster-checks`

  Skip enforcement of cluster-side plugin requirements (Kubernetes/Deckhouse/module versions) when the cluster is unreachable. Defaults to $D8_PLUGINS_SKIP_CLUSTER_CHECKS.
- `--source`

  Pull plugins directly from this registry repo instead of the registry-packages-proxy (bypasses the cluster).
- `--source-login`

  Source registry login. Defaults to $D8_MIRROR_SOURCE_LOGIN.
- `--source-password`

  Source registry password. Defaults to $D8_MIRROR_SOURCE_PASSWORD.
- `--tls-skip-verify`

  Disable TLS certificate validation for the source registry.

d8 dist status

Usage:

```
d8 dist status [options]
```

Show the distribution status: the d8 version, plugins, what is outdated

Show more...

Show the state of the d8 distribution: the running deckhouse-cli version and the
installed plugins, with update status from the registry-packages-proxy.

When the cluster is unreachable, prints the local data and a warning instead of failing.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist update

Usage:

```
d8 dist update [options]
```

Update deckhouse-cli to the latest version

Options

- `--version`

  Exact version to install; downgrades are allowed (default: the latest).

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.

- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist use

Usage:

```
d8 dist use <version> [options]
```

Switch to a specific deckhouse-cli version (no download when it is installed locally)

Show more...

Switch the d8 binary to the given version.

Versions live in the local store (~/.deckhouse-cli/cli/versions) and the active one is
selected by the store's 'current' symlink - the same layout plugins use. Switching to an
installed version repoints that symlink: instant, offline, no elevated privileges.
A version missing from the store is downloaded through the registry-packages-proxy first
(kubeconfig required) and stays installed afterwards.

The first switch migrates a plain-file install to the symlink layout; the original binary
is kept with a ".old" suffix.

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 dist versions

Alias: `list`

Usage:

```
d8 dist versions [options]
```

List deckhouse-cli versions available in the cluster registry

Global options

- `--context`

  Kubeconfig context to use. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements.
- `--insecure-skip-tls-verify`

  Skip TLS verification of both the Kubernetes API server and registry-packages-proxy certificates. For

debugging only.
- `--kubeconfig`, `-k`

  Path to the kubeconfig file. Used to reach registry-packages-proxy and to enforce cluster-side plugin requirements. Defaults to $KUBECONFIG.
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.

d8 help

Help about any command

Show more...

Help provides help for any command in the application.
Simply type d8 help [path to command] for full details.

d8 iam

Manage Deckhouse users, groups, and access grants

Show more...

Manage Deckhouse identity and access: users, groups, and access grants.

Subcommands: user — local users (user-authn Dex): create/delete/reset-password/ reset2fa/lock/unlock. group — local groups: create/delete and add-member/remove-member. access — grant and revoke (current authz model). get — show one user/group/rule with effective access and warnings. list — list users/groups/rules with effective access.

All subcommands accept the standard --kubeconfig / --context flags.

© Flant JSC 2026

d8 iam access

Usage:

```
d8 iam access [options]
```

Grant or revoke access (current authz model)

Show more...

Grant and revoke access in Deckhouse (current authz model).

For inspecting existing rules and effective access (warnings, group cycles, manual rules), use the top-level "d8 iam get" / "d8 iam list".

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam access grant

Usage:

```
d8 iam access grant (user|group) NAME --access-level LEVEL [scope] [options]
```

Grant access to a user or group (current authz model)

Show more...

Grant access to a user or group using the current authorization model.

For users, the subject is resolved by reading User.spec.email from the cluster, because the current authz model requires email as the subject name when user-authn is used.

Specify the scope via -n/--namespace OR --scope (mutually exclusive):

-n/--namespace NS AuthorizationRule per namespace. Repeat to target several namespaces. User, PrivilegedUser, Editor, Admin levels only. --scope cluster ClusterAuthorizationRule, every namespace EXCEPT system ones (d8-*, kube-system). --scope all-namespaces ClusterAuthorizationRule with matchAny: true. Covers ALL namespaces, including system ones. --scope labels=K=V[,...] ClusterAuthorizationRule with namespaceSelector.labelSelector.matchLabels.

--allow-scale and --port-forwarding add capabilities to any scope.

© Flant JSC 2026

Options

- `--access-level`

  Access level: User, PrivilegedUser, Editor, Admin, ClusterEditor, ClusterAdmin, SuperAdmin
- `--allow-scale`

  Allow scaling workloads
- `--dry-run`

  Print the resource(s) that would be created without applying
- `--namespace`, `-n`

  Target namespace(s) for namespaced grants (repeatable). Mutually exclusive with --scope
- `--output`, `-o`

  Output format: name|yaml|json
- `--port-forwarding`

  Allow port-forwarding
- `--scope`

  Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with -n/--namespace

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam access revoke

Usage:

```
d8 iam access revoke (user|group) NAME --access-level LEVEL [scope] [options]
```

Revoke a d8-managed access grant

Show more...

Revoke access previously granted with "d8 iam access grant".

The flags must match the original grant's scope/level — the d8-managed object name is derived from the canonical spec, so a mismatch yields a "not found" error rather than removing an unrelated rule.

Only d8-managed objects (label app.kubernetes.io/managed-by=d8-cli) are affected. Manually maintained AuthorizationRules and ClusterAuthorizationRules are never touched; edit them with kubectl.

© Flant JSC 2026

Options

- `--access-level`

  Access level to revoke
- `--allow-scale`

  Match grants with allow-scale enabled
- `--dry-run`

  Resolve and print the object(s) that would be deleted without touching the cluster
- `--namespace`, `-n`

Target namespace(s). Mutually exclusive with --scope

- `--output`, `-o`

  Output format: name|yaml|json
- `--port-forwarding`

  Match grants with port-forwarding enabled
- `--scope`

  Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with -n/--namespace

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam get

Usage:

```
d8 iam get (user|group|rule) NAME [options]
```

Show a single Deckhouse IAM object

Show more...

Show a single Deckhouse IAM object with its effective context.

Subcommands: user<name> show effective access for a user (groups, grants, summary) group<name> show effective access for a group (members, grants, summary) rule REF show one ClusterAuthorizationRule or AuthorizationRule

Aliases users / groups / rules are accepted for symmetry with "d8 iam list".

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam get group

Alias: `groups`

Usage:

```
d8 iam get (user|group|rule) NAME group <name> [options]
```

Show detailed access for a specific group

Options

- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam get rule

Alias: `rules`

Usage:

```
d8 iam get (user|group|rule) NAME rule REF [options]
```

Show a single CAR or AR with subjects, scope and reverse CR lookup

Options

- `--output`, `-o`

  Output format: text|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam get user

Alias: `users`

Usage:

```
d8 iam get (user|group|rule) NAME user <name> [options]
```

Show detailed access for a specific user

Options

- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam group

Alias: `groups`

Usage:

```
d8 iam group [options]
```

Manage Deckhouse local groups (user-authn)

Show more...

Manage Deckhouse local groups (user-authn).

This command provides lifecycle operations for local Group CRs: Create, Delete, and membership management via add-member / remove-member.

For viewing groups (single or list with effective access), use the top-level commands:

d8 iam get group <name>
d8 iam list groups

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam group add-member

Usage:

```
d8 iam group add-member GROUP [user|group] MEMBER [options]
```

Add a member (user or group) to a local group

Options

- `--create-group`
  Create the target group if it does not exist

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam group create

Usage:

```
d8 iam group create <name> [options]
```

Create a local group in Deckhouse

Options

- `--dry-run`
  Print the resource that would be created without applying
- `--output`, `-o`
  Output format: name|yaml|json

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam group delete

Usage:

```
d8 iam group delete <name> [options]
```

Delete a local group

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam group remove-member

Usage:

```
d8 iam group remove-member GROUP [user|group] MEMBER [options]
```

Remove a member (user or group) from a local group

Global options

- `--context`

The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam list

Usage:

```
d8 iam list (users|groups|rules) [options]
```

List Deckhouse IAM objects

Show more...

List Deckhouse IAM objects with their effective context.

Subcommands: users list every user with effective-access summary (groups, grants count, level) groups list every group with effective-access summary (members, grants count, level) rules list every ClusterAuthorizationRule and AuthorizationRule

Aliases user / group / rule are accepted so both forms work as muscle memory.

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam list groups

Alias: `group`

Usage:

```
d8 iam list (users|groups|rules) groups [options]
```

List all groups with their effective access

Options

- `--output`, `-o`

  Output format: table|json

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam list rules

Alias: `rule`

Usage:

```
d8 iam list (users|groups|rules) rules [options]
```

List ClusterAuthorizationRules and AuthorizationRules

Options

- `--cluster`

  Only ClusterAuthorizationRules
- `--managed-only`

  Only rules managed by d8-cli (label app.kubernetes.io/managed-by=d8-cli)

- `--manual-only`

  Only rules NOT managed by d8-cli
- `--namespace`, `-n`

  Restrict to AuthorizationRules in these namespaces (repeatable). Implicitly disables CARs unless --cluster is also set.
- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam list users

Alias: `user`

Usage:

```
d8 iam list (users|groups|rules) users [options]
```

List all users with their effective access

Options

- `--output`, `-o`

  Output format: table|json

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user

Alias: `userop`

Usage:

```
d8 iam user [options]
```

Manage Deckhouse users (user-authn)

Show more...

Manage Deckhouse local static users (user-authn).

This command provides lifecycle operations for Dex local users: Create, Delete, ResetPassword, Reset2FA, Lock, Unlock.

For viewing users (single or list with effective access), use the top-level commands:

d8 iam get user <name>
d8 iam list users

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user create

Usage:

```
d8 iam user create <name> --email <email> [flags] [options]
```

Create a local static user in Deckhouse

Show more...

Create a local static user in Deckhouse (User CR).

The password can be provided interactively (--password-prompt), piped from stdin (--password-stdin), auto-generated (--generate-password), or supplied as a pre-computed bcrypt hash (--password-hash). If no password flag is given and stdin is a terminal, the command prompts interactively.

When --generate-password is used, the password is shown exactly once on stderr.

If the cluster has a password policy enabled, the user may be required to change the password on first login. If staticUsers2FA is enabled, the user will also need to enroll TOTP on first login.

© Flant JSC 2026

Options

- `--create-groups`

  Create groups specified by --member-of if they do not exist
- `--dry-run`

  Print the resource that would be created without applying
- `--email`

  User email address (required, must be lowercase)
- `--generate-password`

  Auto-generate a strong password (shown once on stderr)
- `--member-of`

  Add user to these groups (repeatable)
- `--output`, `-o`

  Output format: name|yaml|json
- `--password-hash`

  Use a pre-computed bcrypt hash directly (must start with $2 and be a valid bcrypt cost)
- `--password-prompt`

  Read password interactively with hidden input
- `--password-stdin`

  Read password from stdin (for CI/pipelines)
- `--ttl`

  User time-to-live (e.g. 24h, 30m). Can only be set once; expireAt will not update on change.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user delete

Usage:

```
d8 iam user delete <name> [options]
```

Delete a local static user

Show more...

Delete a local static user from Deckhouse.

The User CR is removed and, by default, the user is also removed from every local Group CR that lists them under spec.members. Pass --keep-memberships to leave the membership references in place (e.g. for audit retention or to re-create the user with the same memberships later).

© Flant JSC 2026

Options

- `--keep-memberships`

  Do not remove the user from local Group spec.members; only delete the User CR

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user lock

Usage:

```
d8 iam user lock <username> <lockDuration> [options]
```

Lock local user in Dex for a period of time

Show more...

Lock local user in Dex for a period of time.

The lockDuration argument must be a duration string (e.g. 30s, 10m, 1h).

Options

- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user reset-password

Alias: `resetpass`

Usage:

```
d8 iam user reset-password <username> [options]
```

Reset local user's password in Dex

Show more...

Reset a local static user's password in Dex.

The new password is read from one of:

--password-prompt (default if stdin is a terminal) --password-stdin (echo "newpw" | d8 iam user reset-password ...) --generate-password (auto-generate, shown once on stderr) --password-hash STRING (pre-computed bcrypt hash, must start with $2)

This is the same flag layout as 'd8 iam user create', so muscle memory and shell autocompletion transfer between the two commands.

Internally a UserOperation of type ResetPassword is created with the raw bcrypt hash in spec.resetPassword.newPasswordHash. The user-authn hook validates the prefix and bcrypt.Cost, base64-encodes the hash, and applies it to the Dex Password CR. We never write the plaintext to the API.

© Flant JSC 2026

Options

- `--generate-password`

  Auto-generate a strong password (shown once on stderr)
- `--password-hash`

  Use a pre-computed bcrypt hash directly (must start with $2 and be a valid bcrypt cost)
- `--password-prompt`

  Read password interactively with hidden input
- `--password-stdin`

  Read password from stdin (for CI/pipelines)
- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user reset2fa

Usage:

```
d8 iam user reset2fa <username> [options]
```

Reset local user's 2FA (TOTP) in Dex

Show more...

Reset local user's 2FA (TOTP) in Dex.

This requests a UserOperation of type Reset2FA and waits for completion by default.

Options

- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 iam user unlock

Usage:

```
d8 iam user unlock <username> [options]
```

Unlock local user in Dex

Show more...

Unlock local user in Dex.

This requests a UserOperation of type Unlock and waits for completion by default.

Options

- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 k

Alias: `kubectl`

kubectl controls the Kubernetes cluster manager

Show more...

kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/

Options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k alpha

Usage:

```
d8 k alpha [options]
```

Commands for features in alpha

Show more...

These commands correspond to alpha features that are not enabled in Kubernetes clusters by default.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k alpha kuberc

Usage:

```
d8 k alpha kuberc SUBCOMMAND [options]
```

Manage kuberc configuration files

Show more...

Manage user preferences (kuberc) file.

The kuberc file allows you to customize your kubectl experience.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log

format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k alpha kuberc set

Usage:

```
d8 k alpha kuberc SUBCOMMAND set --section (defaults|aliases) --command COMMAND [options]
```

Set values in the kuberc configuration

Show more...

Set values in the kuberc configuration file.

Use --section to specify whether to set defaults or aliases.

For defaults: Sets default flag values for kubectl commands. The --command flag should specify only the command (e.g., "get", "create", "set env"), not resources.

For aliases: Creates command aliases with optional default flag values and arguments. Use --prependarg and --appendarg to include resources or other arguments.

Options

- `--appendarg`

  Argument to append to the command (can be specified multiple times, for aliases only)
- `--command`

  Command to configure (e.g., 'get', 'create', 'set env')
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--name`

  Alias name (required for --section=aliases)
- `--option`

  Flag option in the form flag=value (can be specified multiple times)
- `--overwrite`

  Allow overwriting existing entries
- `--prependarg`

  Argument to prepend to the command (can be specified multiple times, for aliases only)
- `--section`

  Section to modify: 'defaults' or 'aliases'

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use

- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k alpha kuberc view

Usage:

```
d8 k alpha kuberc SUBCOMMAND view [options]
```

Display the current kuberc configuration

Show more...

Display the contents of the kuberc file in the specified output format.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false

feature gate or turning off the feature KUBERC=off.

- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k annotate

Usage:

```
d8 k annotate [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-version=version]
[options]
```

Update the annotations on a resource

Show more...

Update the annotations on one or more resources.

All Kubernetes objects support the ability to store additional data with the object as annotations. Annotations are key/value pairs that can be larger than labels and include arbitrary string values such as structured JSON. Tools and system extensions may use annotations to store their own data.

Attempting to set an annotation that already exists will fail unless --overwrite is set. If --resource-version is specified and does not match the current resource version on the server the command will fail.

Use "kubectl api-resources" for a complete list of supported resources.

Options

- `--all`

  Select all resources, in the namespace of the specified resource types.
- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update the annotation
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the annotations for a given resource.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile,

jsonpath, jsonpath-as-json, jsonpath-file).

- `--overwrite`

  If true, allow annotations to be overwritten, otherwise reject annotation updates that overwrite existing annotations.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the annotation update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-resources

Usage:

```
d8 k api-resources [options]
```

Print the supported API resources on the server

Show more...

Print the supported API resources on the server.

Options

- `--api-group`

  Limit to resources in the specified API group.
- `--cached`

  Use the cached list of resources if available.
- `--categories`

  Limit to resources that belong to the specified categories.
- `--namespaced`

  If false, non-namespaced resources will be returned, otherwise returning namespaced resources by default.
- `--no-headers`

  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, wide).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--sort-by`

  If non-empty, sort list of resources using specified field. The field can be either 'name' or 'kind'.

- `--verbs`

  Limit to resources that support the specified verbs.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-versions

Usage:

```
d8 k api-versions [options]
```

Print the supported API versions on the server, in the form of "group/version"

Show more...

Print the supported API versions on the server, in the form of "group/version".

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version

- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) [options]
```

Apply a configuration to a resource by file name or stdin

Show more...

Apply a configuration to a resource by file name or stdin. The resource name must be specified. This resource will be created if it doesn't exist yet. To use 'apply', always create the resource initially with either 'apply' or 'create --save-config'.

JSON and YAML formats are accepted.

Alpha Disclaimer: the --prune functionality is not yet complete. Do not use unless you are aware of what the current state is. See https://issues.k8s.io/34274.

Options

- `--all`

  Select all resources in the namespace of the specified resource types.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.

- `--filename`, `-f`

  The files that contain the configurations to apply.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--force-conflicts`

  If true, server-side apply will force the changes against conflicts.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--openapi-patch`

  If true, use openapi to calculate diff when the openapi presents and the resource can be found in the openapi spec. Otherwise, fall back to use baked-in types.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  Automatically resolve conflicts between the modified and live configuration by using values from the modified configuration
- `--prune`

  Automatically delete resource objects, that do not appear in the configs and are created by either apply or create --save-config. Should be used with either -l or --all.
- `--prune-allowlist`

  Overwrite the default allowlist with for --prune
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-side`

  If true, apply runs in the server instead of the client.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, apply will operate on the subresource of the requested object. Only allowed when using --server-side.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply edit-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) edit-last-applied (RESOURCE/NAME | -f FILENAME) [options]
```

Edit latest last-applied-configuration annotations of a resource/object

Show more...

Edit the latest last-applied-configuration annotations of resources from the default editor.

The edit-last-applied command allows you to directly edit any API resource you can retrieve via the command-line tools. It will open the editor defined by your KUBE_EDITOR, or EDITOR environment variables, or fall back to 'vi' for Linux or 'notepad' for Windows. You can edit multiple objects, although changes are applied one at a time. The command accepts file names as well as command-line arguments, although the files you point to must be previously saved versions of resources.

The default format is YAML. To edit in JSON, specify "-o json".

The flag --windows-line-endings can be used to force Windows line endings, otherwise the default for your operating system will be used.

In the event an error occurs while updating, a temporary file will be created on disk that contains your unapplied changes. The most common error when updating a resource is another editor changing the resource on the server. When this occurs, you will have to apply your changes to the newer version of the resource, or update your temporary saved copy to include the latest resource version.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files to use to edit the resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn

about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--windows-line-endings`

  Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the

server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply set-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) set-last-applied -f FILENAME [options]
```

Set the last-applied-configuration annotation on a live object to match the contents of a file

Show more...

Set the latest last-applied-configuration annotations by setting it to match the contents of a file. This results in the last-applied-configuration being updated as though 'kubectl apply -f<file> ' was run, without updating any other parts of the object.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--create-annotation`

  Will create 'last-applied-configuration' annotations if current objects doesn't have one
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files that contains the last-applied-configuration annotations
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory

- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply view-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) view-last-applied (TYPE [NAME | -l label] | TYPE/NAME | -f FILENAME)
[options]
```

View the latest last-applied-configuration annotations of a resource/object

Show more...

View the latest last-applied-configuration annotations by type/name or file.

The default output will be printed to stdout in YAML format. You can use the -o option to change the output format.

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--filename`, `-f`

  Filename, directory, or URL to files that contains the last-applied-configuration annotations
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. Must be one of (yaml, json)
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k attach

Usage:

```
d8 k attach (POD | TYPE/NAME) -c CONTAINER [options]
```

Attach to a running container

Show more...

Attach to a process that is already running inside an existing container.

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--quiet`, `-q`

  Only print output from the remote session
- `--stdin`, `-i`

  Pass stdin to the container
- `--tty`, `-t`

  Stdin is a TTY

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth

Usage:

```
d8 k auth [options]
```

Inspect authorization

Show more...

Inspect authorization.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth can-i

Usage:

```
d8 k auth can-i VERB [TYPE | TYPE/NAME | NONRESOURCEURL] [options]
```

Check whether an action is allowed

Show more...

Check whether an action is allowed.

VERB is a logical Kubernetes API verb like 'get', 'list', 'watch', 'delete', etc. TYPE is a Kubernetes resource. Shortcuts and groups will be resolved. NONRESOURCEURL is a partial URL that starts with "/". NAME is the name of a particular Kubernetes resource. This command pairs nicely with impersonation. See --as global flag.

Options

- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.
- `--list`

  If true, prints all allowed actions.
- `--no-headers`

  If true, prints allowed actions without headers
- `--quiet`, `-q`

  If true, suppress output and just return the exit code.
- `--subresource`

  SubResource such as pod/log or deployment/scale

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth reconcile

Usage:

```
d8 k auth reconcile -f FILENAME [options]
```

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects

Show more...

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects.

Missing objects are created, and the containing namespace is created for namespaced objects, if required.

Existing roles are updated to include the permissions in the input objects, and remove extra permissions if --remove-extra-permissions is specified.

Existing bindings are updated to include the subjects in the input objects, and remove extra subjects if --remove-extra-subjects is specified.

This is preferred to 'apply' for RBAC resources so that semantically-aware merging of rules and subjects is done.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to reconcile.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--remove-extra-permissions`

  If true, removes extra permissions added to roles
- `--remove-extra-subjects`

  If true, removes extra subjects added to rolebindings
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth whoami

Usage:

```
d8 k auth whoami [options]
```

Experimental: Check self subject attributes

Show more...

Experimental: Check who you are and your attributes (groups, extra).

This command is helpful to get yourself aware of the current user attributes,
especially when dynamic authentication, e.g., token webhook, auth proxy, or OIDC provider,
is enabled in the Kubernetes cluster.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request

- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k autoscale

Usage:

```
d8 k autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu=CPU] [--memory=MEMORY]
[options]
```

Auto-scale a deployment, replica set, stateful set, or replication controller

Show more...

Creates an autoscaler that automatically chooses and sets the number of pods that run in a Kubernetes cluster. The command will attempt to use the autoscaling/v2 API first, in case of an error, it will fall back to autoscaling/v1 API.

Looks up a deployment, replica set, stateful set, or replication controller by name and creates an autoscaler that uses the given resource as a reference. An autoscaler can automatically increase or decrease number of pods deployed within the system as needed.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cpu`

  Target CPU utilization over all the pods. When specified as a percentage (e.g."70%" for 70% of requested CPU) it will target average utilization. When specified as quantity (e.g."500m" for 500 milliCPU) it will target average value. Value without units is treated as a quantity with miliCPU being the unit (e.g."500" is "500m").
- `--cpu-percent`

  The target average CPU utilization (represented as a percent of requested CPU) over all the pods. If it's not specified or negative, a default autoscaling policy will be used.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

  Filename, directory, or URL to files identifying the resource to autoscale.
- `--kustomize, -k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--max`

  The upper limit for the number of pods that can be set by the autoscaler. Required.
- `--memory`

  Target memory utilization over all the pods. When specified as a percentage (e.g."60%" for 60% of requested memory) it will target average utilization. When specified as quantity (e.g."200Mi" for 200 MiB, "1Gi" for 1 GiB) it will target average value. Value without units is treated as a quantity with mebibytes being the unit (e.g."200" is "200Mi").
- `--min`

  The lower limit for the number of pods that can be set by the autoscaler. If it's not specified or negative, the server will apply a default value.
- `--name`

  The name for the newly created object. If not specified, the name of the input resource will be used.
- `--output, -o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate

Usage:

```
d8 k certificate SUBCOMMAND [options]
```

Modify certificate resources

Show more...

Modify certificate resources.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate approve

Usage:

```
d8 k certificate SUBCOMMAND approve (-f FILENAME | NAME) [options]
```

Approve a certificate signing request

Show more...

Approve a certificate signing request.

kubectl certificate approve allows a cluster admin to approve a certificate signing request (CSR). This action tells a certificate signing controller to issue a certificate to the requester with the attributes requested in the CSR.

SECURITY NOTICE: Depending on the requested attributes, the issued certificate can potentially grant a requester access to cluster resources or to authenticate as a requested identity. Before approving a CSR, ensure you understand what the signed certificate can do.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--force`

  Update the CSR even if it is already approved.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate deny

Usage:

```
d8 k certificate SUBCOMMAND deny (-f FILENAME | NAME) [options]
```

Deny a certificate signing request

Show more...

Deny a certificate signing request.

kubectl certificate deny allows a cluster admin to deny a certificate signing request (CSR). This action tells a certificate signing controller to not to issue a certificate to the requester.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--force`

  Update the CSR even if it is already denied.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info

Usage:

```
d8 k cluster-info [options]
```

Display cluster information

Show more...

Display addresses of the control plane and services with label kubernetes.io/cluster-service=true. To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info dump

Usage:

```
d8 k cluster-info dump [options]
```

Dump relevant information for debugging and diagnosis

Show more...

Dump cluster information out suitable for debugging and diagnosing cluster problems. By default, dumps everything to stdout. You can optionally specify a directory with --output-directory. If you specify a directory, Kubernetes will build a set of files in that directory. By default, only dumps things in the current namespace and 'kube-system' namespace, but you can switch to a different namespace with the --namespaces flag, or specify --all-namespaces to dump all namespaces.

The command also dumps the logs of all of the pods in the cluster; these logs are dumped into different directories based on namespace and pod name.

Options

- `--all-namespaces`, `-A`

  If true, dump all namespaces. If true, --namespaces is ignored.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--namespaces`

  A comma separated list of namespaces to dump.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--output-directory`

  Where to output the files. If empty or '-' uses stdout, otherwise creates a directory hierarchy in that directory
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k completion

Usage:

```
d8 k completion SHELL [options]
```

Output shell completion code for the specified shell (bash, zsh, fish, or powershell)

Show more...

Output shell completion code for the specified shell (bash, zsh, fish, or powershell). The shell code must be evaluated to provide interactive completion of kubectl commands. This can be done by sourcing it from the .bash_profile.

Detailed instructions on how to do this are available here:

for macOS:
https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/#enable-shell-autocompletion

for linux:
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#enable-shell-autocompletion

for windows:
https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/#enable-shell-autocompletion

Note for zsh users: [1] zsh completions are only supported in versions of zsh >= 5.2.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config

Usage:

```
d8 k config SUBCOMMAND [options]
```

Modify kubeconfig files

Show more...

Modify kubeconfig files using subcommands like "kubectl config set current-context my-context".

The loading order follows these rules:

1. If the --kubeconfig flag is set, then only that file is loaded. The flag may only be set once and no merging takes place.
2. If $KUBECONFIG environment variable is set, then it is used as a list of paths (normal path delimiting rules for your system). These paths are merged. When a value is modified, it is modified in the file that defines the stanza. When a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the last file in the list.
3. Otherwise, ${HOME}/.kube/config is used and no merging takes place.

Options

- `--kubeconfig`

  use a particular kubeconfig file

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config current-context

Usage:

```
d8 k config SUBCOMMAND current-context [options]
```

Display the current-context

Show more...

Display the current-context.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory

- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-cluster

Usage:

```
d8 k config SUBCOMMAND delete-cluster NAME [options]
```

Delete the specified cluster from the kubeconfig

Show more...

Delete the specified cluster from the kubeconfig.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-context

Usage:

```
d8 k config SUBCOMMAND delete-context NAME [options]
```

Delete the specified context from the kubeconfig

Show more...

Delete the specified context from the kubeconfig.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-user

Usage:

```
d8 k config SUBCOMMAND delete-user NAME [options]
```

Delete the specified user from the kubeconfig

Show more...

Delete the specified user from the kubeconfig.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-clusters

Usage:

```
d8 k config SUBCOMMAND get-clusters [options]
```

Display clusters defined in the kubeconfig

Show more...

Display clusters defined in the kubeconfig.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server

- `--v`, `-v`
  number for the log level verbosity
- `--vmodule`
  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-contexts

Usage:

```
d8 k config SUBCOMMAND get-contexts [(-o|--output=)name)] [options]
```

Describe one or many contexts

Show more...

Display one or many contexts from the kubeconfig file.

Options

- `--no-headers`
  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`
  Output format. One of: (name).

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-users

Usage:

```
d8 k config SUBCOMMAND get-users [options]
```

Display users defined in the kubeconfig

Show more...

Display users defined in the kubeconfig.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config rename-context

Usage:

```
d8 k config SUBCOMMAND rename-context CONTEXT_NAME NEW_NAME [options]
```

Rename a context from the kubeconfig file

Show more...

Renames a context from the kubeconfig file.

CONTEXT_NAME is the context name that you want to change.

NEW_NAME is the new name you want to set.

Note: If the context being renamed is the 'current-context', this field will also be updated.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set

Usage:

```
d8 k config SUBCOMMAND set PROPERTY_NAME PROPERTY_VALUE [options]
```

Set an individual value in a kubeconfig file

Show more...

Set an individual value in a kubeconfig file.

PROPERTY_NAME is a dot delimited name where each token represents either an attribute name or a map key. Map keys may not contain dots.

PROPERTY_VALUE is the new value you want to set. Binary fields such as 'certificate-authority-data' expect a base64 encoded string unless the --set-raw-bytes flag is used.

Specifying an attribute name that already exists will merge new fields on top of existing values.

Options

- `--set-raw-bytes`

  When writing a []byte PROPERTY_VALUE, write the given string directly without base64 decoding.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-cluster

Usage:

```
d8 k config SUBCOMMAND set-cluster NAME [--server=server] [--certificate-authority=path/to/certificate/authority]
[--insecure-skip-tls-verify=true] [--tls-server-name=example.com] [options]
```

Set a cluster entry in kubeconfig

Show more...

Set a cluster entry in kubeconfig.

Specifying a name that already exists will merge new fields on top of existing values for those fields.

Options

- `--certificate-authority`

  Path to certificate-authority file for the cluster entry in kubeconfig
- `--embed-certs`

  embed-certs for the cluster entry in kubeconfig

- `--insecure-skip-tls-verify`

  insecure-skip-tls-verify for the cluster entry in kubeconfig
- `--proxy-url`

  proxy-url for the cluster entry in kubeconfig
- `--server`

  server for the cluster entry in kubeconfig
- `--tls-server-name`

  tls-server-name for the cluster entry in kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server

- `--v`, `-v`
  number for the log level verbosity
- `--vmodule`
  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-context

Usage:

```
d8 k config SUBCOMMAND set-context [NAME | --current] [--cluster=cluster_nickname] [--user=user_nickname] [--
namespace=namespace] [options]
```

Set a context entry in kubeconfig

Show more...

Set a context entry in kubeconfig.

Specifying a name that already exists will merge new fields on top of existing values for those fields.

Options

- `--cluster`
  cluster for the context entry in kubeconfig
- `--current`
  Modify the current context
- `--namespace`, `-n`
  namespace for the context entry in kubeconfig
- `--user`
  user for the context entry in kubeconfig

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-credentials

Usage:

```
d8 k config SUBCOMMAND set-credentials NAME [--client-certificate=path/to/certfile] [--client-key=path/to/keyfile]
[--token=bearer_token] [--username=basic_user] [--password=basic_password] [--auth-provider=provider_name] [--auth-
provider-arg=key=value] [--exec-command=exec_command] [--exec-api-version=exec_api_version] [--exec-arg=arg] [--
exec-env=key=value] [options]
```

Set a user entry in kubeconfig

Show more...

Set a user entry in kubeconfig.

Specifying a name that already exists will merge new fields on top of existing values.

Client-certificate flags:
--client-certificate=certfile --client-key=keyfile

Bearer token flags:
--token=bearer_token

Basic auth flags:
--username=basic_user --password=basic_password

Bearer token and basic auth are mutually exclusive.

Options

- `--auth-provider`

  Auth provider for the user entry in kubeconfig
- `--auth-provider-arg`

  'key=value' arguments for the auth provider
- `--client-certificate`

Path to client-certificate file for the user entry in kubeconfig
- `--client-key`

  Path to client-key file for the user entry in kubeconfig
- `--embed-certs`

  Embed client cert/key for the user entry in kubeconfig
- `--exec-api-version`

  API version of the exec credential plugin for the user entry in kubeconfig
- `--exec-arg`

  New arguments for the exec credential plugin command for the user entry in kubeconfig
- `--exec-command`

  Command for the exec credential plugin for the user entry in kubeconfig
- `--exec-env`

  'key=value' environment values for the exec credential plugin
- `--exec-interactive-mode`

  InteractiveMode of the exec credentials plugin for the user entry in kubeconfig
- `--exec-provide-cluster-info`

  ProvideClusterInfo of the exec credentials plugin for the user entry in kubeconfig
- `--password`

  password for the user entry in kubeconfig
- `--token`

  token for the user entry in kubeconfig
- `--username`

  username for the user entry in kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--user`

  The name of the kubeconfig user to use
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config unset

Usage:

```
d8 k config SUBCOMMAND unset PROPERTY_NAME [options]
```

Unset an individual value in a kubeconfig file

Show more...

Unset an individual value in a kubeconfig file.

PROPERTY_NAME is a dot delimited name where each token represents either an attribute name or a map key. Map keys may not contain dots.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config use-context

Alias: `use`

Usage:

```
d8 k config SUBCOMMAND use-context CONTEXT_NAME [options]
```

Set the current-context in a kubeconfig file

Show more...

Set the current-context in a kubeconfig file.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log

format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config view

Usage:

```
d8 k config SUBCOMMAND view [options]
```

Display merged kubeconfig settings or a specified kubeconfig file

Show more...

Display merged kubeconfig settings or a specified kubeconfig file.

You can use --output jsonpath={...} to extract specific values using a jsonpath expression.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--flatten`

  Flatten the resulting kubeconfig file into self-contained output (useful for creating portable kubeconfig files)
- `--merge`

  Merge the full hierarchy of kubeconfig files
- `--minify`

  Remove all information not used by current-context from the output
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--raw`

  Display raw byte data and sensitive data
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use

- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cordon

Usage:

```
d8 k cordon NODE [options]
```

Mark node as unschedulable

Show more...

Mark node as unschedulable.

Options

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--selector`, `-l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cp

Usage:

```
d8 k cp <file-spec-src> <file-spec-dest> [options]
```

Copy files and directories to and from containers

Show more...

Copy files and directories to and from containers.

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--no-preserve`

  The copied file/directory's ownership and permissions will not be preserved in the container
- `--retries`

  Set number of retries to complete a copy operation from a container. Specify 0 to disable or any negative value for infinite retrying. The default is 0 (no retry).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections

insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create

Usage:

```
d8 k create -f FILENAME [options]
```

Create a resource from a file or from stdin

Show more...

Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--edit`

Edit the API resource before creating
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename, -f`

  Filename, directory, or URL to files to use to create the resource
- `--kustomize, -k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--raw`

  Raw URI to POST to the server. Uses the transport specified by the kubeconfig file.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector, -l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--windows-line-endings`

  Only relevant if --edit=true. Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrole

Usage:

```
d8 k create -f FILENAME clusterrole NAME --verb=verb --resource=resource.group [--resource-name=resourcename] [--
dry-run=server|client|none] [options]
```

Create a cluster role

Show more...

Create a cluster role.

Options

- `--aggregation-rule`

  An aggregation label selector for combining ClusterRoles.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--non-resource-url`

  A partial url that user should have access to.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`

  Resource that the rule applies to
- `--resource-name`

  Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`

  Verb that applies to the resources contained in the rule

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrolebinding

Usage:

```
d8 k create -f FILENAME clusterrolebinding NAME --clusterrole=NAME [--user=username] [--group=groupname] [--
serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a cluster role binding for a particular cluster role

Show more...

Create a cluster role binding for a particular cluster role.

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--clusterrole`

  ClusterRole this ClusterRoleBinding should reference
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--group`

  Groups to bind to the clusterrole. The flag can be repeated to add multiple groups.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--serviceaccount`

  Service accounts to bind to the clusterrole, in the format :. The flag can be repeated to add multiple service accounts.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the clusterrole. The flag can be repeated to add multiple users.
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use

- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create configmap

Alias: `cm`

Usage:

```
d8 k create -f FILENAME configmap NAME [--from-file=[key=]source] [--from-literal=key1=value1] [--dry-
run=server|client|none] [options]
```

Create a config map from a local file, directory or literal value

Show more...

Create a config map based on a file, directory, or specified literal value.

A single config map may package one or more key/value pairs.

When creating a config map based on a file, the key will default to the basename of the file, and the value will default to the file content. If the basename is an invalid key, you may specify an alternate key.

When creating a config map based on a directory, each file whose basename is a valid key in the directory will be packaged into the config map. Any directory entries except regular files are ignored (e.g. subdirectories,

symlinks, devices, pipes, etc).

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the configmap to its name.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-env-file`

  Specify the path to a file to read lines of key=val pairs to create a configmap.
- `--from-file`

  Key file can be specified using its file path, in which case file basename will be used as configmap key, or optionally with a key and file path, in which case the given key will be used. Specifying a directory will iterate each named file in the directory whose basename is a valid configmap key.
- `--from-literal`

  Specify a key and literal value to insert in configmap (i.e. mykey=somevalue)
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create cronjob

Alias: `cj`

Usage:

```
d8 k create -f FILENAME cronjob NAME --image=image --schedule='0/5 * * * ?' -- [COMMAND] [args...] [options]
```

Create a cron job with the specified name

Show more...

Create a cron job with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--image`

  Image name to run.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--restart`

  job's restart policy. supported values: OnFailure, Never
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--schedule`

  A schedule in the Cron format the job should be run with.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create deployment

Alias: `deploy`

Usage:

```
d8 k create -f FILENAME deployment NAME --image=image -- [COMMAND] [args...] [options]
```

Create a deployment with the specified name

Show more...

Create a deployment with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

  Name of the manager used to track field ownership.
- `--image`

  Image names to run. A deployment can have multiple images set for multi-container pod.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--port`

  The containerPort that this deployment exposes.
- `--replicas`, `-r`

  Number of replicas to create. Default is 1.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create ingress

Alias: `ing`

Usage:

```
d8 k create -f FILENAME ingress NAME --rule=host/path=service:port[,tls[=secret]]  [options]
```

Create an ingress with the specified name

Show more...

Create an ingress with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--annotation`

  Annotation to insert in the ingress object, in the format annotation=value
- `--class`

  Ingress Class to be used
- `--default-backend`

  Default service for backend, in format of svcname:port

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--rule`

  Rule in format host/path=service:port[,tls=secretname]. Paths containing the leading character '*' are considered pathType=Prefix. tls argument is optional.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create job

Usage:

```
d8 k create -f FILENAME job NAME --image=image [--from=cronjob/name] -- [COMMAND] [args...] [options]
```

Create a job with the specified name

Show more...

Create a job with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from`

  The name of the resource to create a Job from (only CronJob is supported).
- `--image`

  Image name to run.

- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request

- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create namespace

Alias: `ns`

Usage:

```
d8 k create -f FILENAME namespace NAME [--dry-run=server|client|none] [options]
```

Create a namespace with the specified name

Show more...

Create a namespace with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server

- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create poddisruptionbudget

Alias: `pdb`

Usage:

```
d8 k create -f FILENAME poddisruptionbudget NAME --selector=SELECTOR --min-available=N [--dry-
run=server|client|none] [options]
```

Create a pod disruption budget with the specified name

Show more...

Create a pod disruption budget with the specified name, selector, and desired minimum available pods.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--max-unavailable`

  The maximum number or percentage of unavailable pods this budget requires.
- `--min-available`

  The minimum number or percentage of available pods this budget requires.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`

  A label selector to use for this budget. Only equality-based selector requirements are supported.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn

about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create priorityclass

Alias: `pc`

Usage:

```
d8 k create -f FILENAME priorityclass NAME --value=VALUE --global-default=BOOL [--dry-run=server|client|none]
[options]
```

Create a priority class with the specified name

Show more...

Create a priority class with the specified name, value, globalDefault and description.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--description`

  description is an arbitrary string that usually provides guidelines on when this priority class should be used.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--global-default`

  global-default specifies whether this PriorityClass should be considered as the default priority.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--preemption-policy`

  preemption-policy is the policy for preempting pods with lower priority.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--value`

the value of this priority class.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
  Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create quota

Alias: `resourcequota`

Usage:

```
d8 k create -f FILENAME quota NAME [--hard=key1=value1,key2=value2] [--scopes=Scope1,Scope2] [--dry-
run=server|client|none] [options]
```

Create a quota with the specified name

Show more...

Create a resource quota with the specified name, hard limits, and optional scopes.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--hard`

  A comma-delimited set of resource=quantity pairs that define a hard limit.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--scopes`

  A comma-delimited set of quota scopes that must all match each object tracked by the quota.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log

format)

- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create role

Usage:

```
d8 k create -f FILENAME role NAME --verb=verb --resource=resource.group/subresource [--resource-name=resourcename]
[--dry-run=server|client|none] [options]
```

Create a role with single rule

Show more...

Create a role with single rule.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`

  Resource that the rule applies to
- `--resource-name`

  Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`

  Verb that applies to the resources contained in the rule

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create rolebinding

Usage:

```
d8 k create -f FILENAME rolebinding NAME --clusterrole=NAME|--role=NAME [--user=username] [--group=groupname] [--
serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a role binding for a particular role or cluster role

Show more...

Create a role binding for a particular role or cluster role.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--clusterrole`

  ClusterRole this RoleBinding should reference
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--group`

  Groups to bind to the role. The flag can be repeated to add multiple groups.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--role`

  Role this RoleBinding should reference
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--serviceaccount`

  Service accounts to bind to the role, in the format :. The flag can be repeated to add multiple service accounts.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the role. The flag can be repeated to add multiple users.
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) [options]
```

Create a secret using a specified subcommand

Show more...

Create a secret with specified type.

A docker-registry type secret is for accessing a container registry.

A generic type secret indicate an Opaque secret type.

A tls type secret holds TLS certificate and its associated key.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret docker-registry

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) docker-registry NAME --docker-username=user --docker-password=password --docker-email=email [--docker-server=string] [--from-file=[key=]source] [--dry-run=server|client|none] [options]
```

Create a secret for use with a Docker registry

Show more...

Create a new secret for use with Docker registries.

Dockercfg secrets are used to authenticate against Docker registries.

When using the Docker command line to push images, you can authenticate to a given registry by running: '$ docker login DOCKER_REGISTRY_SERVER --username=DOCKER_USER --password=DOCKER_PASSWORD --email=DOCKER_EMAIL'.

That produces a ~/.dockercfg file that is used by subsequent 'docker push' and 'docker pull' commands to authenticate to the registry. The email address is optional.

When creating applications, you may have a Docker registry that requires authentication. In order for the nodes to pull images on your behalf, they must have the credentials. You can provide this information by creating a dockercfg secret and attaching it to your service account.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--docker-email`

  Email for Docker registry
- `--docker-password`

  Password for Docker registry authentication
- `--docker-server`

  Server location for Docker registry
- `--docker-username`

  Username for Docker registry authentication
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-file`

  Key files can be specified using their file path, in which case a default name of .dockerconfigjson will be given to them, or optionally with a name and file path, in which case the given name will be used.

Specifying a directory will iterate each named file in the directory that is a valid secret key. For this command, the key should always be .dockerconfigjson.

- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request

- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret generic

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) generic NAME [--type=string] [--from-file=
[key=]source] [--from-literal=key1=value1] [--dry-run=server|client|none] [options]
```

Create a secret from a local file, directory, or literal value

Show more...

Create a secret based on a file, directory, or specified literal value.

A single secret may package one or more key/value pairs.

When creating a secret based on a file, the key will default to the basename of the file, and the value will default to the file content. If the basename is an invalid key or you wish to chose your own, you may specify an alternate key.

When creating a secret based on a directory, each file whose basename is a valid key in the directory will be packaged into the secret. Any directory entries except regular files are ignored (e.g. subdirectories, symlinks, devices, pipes, etc).

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-env-file`

Specify the path to a file to read lines of key=val pairs to create a secret.

- `--from-file`

  Key files can be specified using their file path, in which case a default name will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key.
- `--from-literal`

  Specify a key and literal value to insert in secret (i.e. mykey=somevalue)
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

  The type of secret to create
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret tls

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) tls NAME --cert=path/to/cert/file --
key=path/to/key/file [--dry-run=server|client|none] [options]
```

Create a TLS secret

Show more...

Create a TLS secret from the given public/private key pair.

The public/private key pair must exist beforehand. The public key certificate must be .PEM encoded and match the given private key.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--cert`

  Path to PEM encoded public key certificate.
- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

  Name of the manager used to track field ownership.
- `--key`

  Path to private key associated with given certificate.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service

Alias: `svc`

Usage:

```
d8 k create -f FILENAME service [options]
```

Create a service using a specified subcommand

Show more...

Create a service using a specified subcommand.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service clusterip

Usage:

```
d8 k create -f FILENAME service clusterip NAME [--tcp=<port>:<targetPort>] [--dry-run=server|client|none] [options]
```

Create a ClusterIP service

Show more...

Create a ClusterIP service with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--clusterip`

  Assign your own ClusterIP or set to 'None' for a 'headless' service (no loadbalancing).
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use

- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service externalname

Usage:

```
d8 k create -f FILENAME service externalname NAME --external-name external.name [--dry-run=server|client|none]
[options]
```

Create an ExternalName service

Show more...

Create an ExternalName service with the specified name.

ExternalName service references to an external DNS address instead of only pods, which will allow application authors to reference services that exist off platform, on other clusters, or locally.

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--external-name`

  External name of service
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service loadbalancer

Usage:

```
d8 k create -f FILENAME service loadbalancer NAME [--tcp=port:targetPort] [--dry-run=server|client|none] [options]
```

Create a LoadBalancer service

Show more...

Create a LoadBalancer service with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service nodeport

Usage:

```
d8 k create -f FILENAME service nodeport NAME [--tcp=port:targetPort] [--dry-run=server|client|none] [options]
```

Create a NodePort service

Show more...

Create a NodePort service with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--node-port`

  Port used to expose the service on each node in a cluster.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

Port pairs can be specified as ':'.

- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create serviceaccount

Alias: `sa`

Usage:

```
d8 k create -f FILENAME serviceaccount NAME [--dry-run=server|client|none] [options]
```

Create a service account with the specified name

Show more...

Create a service account with the specified name.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create token

Usage:

```
d8 k create -f FILENAME token SERVICE_ACCOUNT_NAME [options]
```

Request a service account token

Show more...

Request a service account token.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--audience`

  Audience of the requested token. If unset, defaults to requesting a token for use with the Kubernetes API server. May be repeated to request a token valid for multiple audiences.
- `--bound-object-kind`

  Kind of an object to bind the token to. Supported kinds are Node, Pod, Secret. If set, --bound-object-name must be provided.
- `--bound-object-name`

  Name of an object to bind the token to. The token will expire when the object is deleted. Requires --bound-object-kind.
- `--bound-object-uid`

  UID of an object to bind the token to. Requires --bound-object-kind and --bound-object-name. If unset, the UID of the existing object is used.
- `--duration`

  Requested lifetime of the issued token. If not set or if set to 0, the lifetime will be determined by the server automatically. The server may return a token with a longer or shorter lifetime.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k debug

Usage:

```
d8 k debug (POD | TYPE[[.VERSION].GROUP]/NAME) [ -- COMMAND [args...] ] [options]
```

Create debugging sessions for troubleshooting workloads and nodes

Show more...

Debug cluster resources using interactive debugging containers.

'debug' provides automation for common debugging tasks for cluster objects identified by resource and name. Pods will be used by default if no resource is specified.

The action taken by 'debug' varies depending on what resource is specified. Supported actions include:

* Workload: Create a copy of an existing pod with certain attributes changed, for example changing the image tag to a new version.
* Workload: Add an ephemeral container to an already running pod, for example to add debugging utilities without restarting the pod.
* Node: Create a new pod that runs in the node's host namespaces and can access the node's filesystem.

Note: When a non-root user is configured for the entire target Pod, some capabilities granted by debug profile may not work.

Options

- `--arguments-only`

  If specified, everything after -- will be passed to the new container as Args instead of Command.
- `--attach`

  If true, wait for the container to start running, and then attach as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true.
- `--container`, `-c`

  Container name to use for debug container.
- `--copy-to`

  Create a copy of the target Pod with this name.
- `--custom`

  Path to a JSON or YAML file containing a partial container spec to customize built-in debug profiles.
- `--env`

  Environment variables to set in the container.
- `--filename`, `-f`

  identifying the resource to debug
- `--image`

  Container image to use for debug container. If not specified, the platform's recommended image will be used.
- `--image-pull-policy`

  The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.
- `--keep-annotations`

  If true, keep the original pod annotations.(This flag only works when used with '--copy-to')
- `--keep-init-containers`

  Run the init containers for the pod. Defaults to true.(This flag only works when used with '--copy-to')
- `--keep-labels`

  If true, keep the original pod labels.(This flag only works when used with '--copy-to')
- `--keep-liveness`

  If true, keep the original pod liveness probes.(This flag only works when used with '--copy-to')
- `--keep-readiness`

  If true, keep the original pod readiness probes.(This flag only works when used with '--copy-to')
- `--keep-startup`

  If true, keep the original startup probes.(This flag only works when used with '--copy-to')
- `--profile`

  Options are "legacy", "general", "baseline", "netadmin", "restricted" or "sysadmin".
- `--quiet`, `-q`

  If true, suppress informational messages.
- `--replace`

  When used with '--copy-to', delete the original Pod.
- `--same-node`

When used with '--copy-to', schedule the copy of target Pod on the same node.

- `--set-image`

  When used with '--copy-to', a list of name=image pairs for changing container images, similar to how 'kubectl set image' works.
- `--share-processes`

  When used with '--copy-to', enable process namespace sharing in the copy.
- `--stdin`, `-i`

  Keep stdin open on the container(s) in the pod, even if nothing is attached.
- `--target`

  When using an ephemeral container, target processes in this container name.
- `--tty`, `-t`

  Allocate a TTY for the debugging container.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a

corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k delete

Usage:

```
d8 k delete ([-f FILENAME] | [-k DIRECTORY] | TYPE [(NAME | -l label | --all)]) [options]
```

Delete resources by file names, stdin, resources and names, or by resources and label selector

Show more...

Delete resources by file names, stdin, resources and names, or by resources and label selector.

JSON and YAML formats are accepted. Only one type of argument may be specified: file names, resources and names, or resources and label selector.

Some resources, such as pods, support graceful deletion. These resources define a default period before they are forcibly terminated (the grace period) but you may override that value with the --grace-period flag, or pass --now to set a grace-period of 1. Because these resources often represent entities in the cluster, deletion may not be acknowledged immediately. If the node hosting a pod is down or cannot reach the API server, termination may take significantly longer than the grace period. To force delete a resource, you must specify the --force flag. Note: only a subset of resources support graceful deletion. In absence of the support, the --grace-period flag is ignored.

IMPORTANT: Force deleting pods does not wait for confirmation that the pod's processes have been terminated, which can leave those processes running until the node detects the deletion and completes graceful deletion. If your processes use shared storage or talk to a remote API and depend on the name of the pod to identify themselves, force deleting those pods may result in multiple processes running on different machines using the same identification which may lead to data corruption or inconsistency. Only force delete pods when you are sure the pod is terminated, or if your application can tolerate multiple copies of the same pod running at once. Also, if you force delete pods, the scheduler may place new pods on those nodes before the node has released those resources and causing those pods to be evicted immediately.

Note that the delete command does NOT do resource version checks, so if someone submits an update to a resource right when you submit a delete, their update will be lost along with the rest of the resource.

After a CustomResourceDefinition is deleted, invalidation of discovery cache may take up to 6 hours. If you don't want to wait, you might want to run "kubectl api-resources" to refresh the discovery cache.

Options

- `--all`

  Delete all resources, in the namespace of the specified resource types.
- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without

sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  containing the resource to delete.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--ignore-not-found`

  Treat "resource not found" as a successful delete. Defaults to "true" when --all is specified.
- `--interactive`, `-i`

  If true, delete resource only when user confirms.
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--now`

  If true, resources are signaled for immediate shutdown (same as --grace-period=1).
- `--output`, `-o`

  Output mode. Use "-o name" for shorter output (resource/name).
- `--raw`

  Raw URI to DELETE to the server. Uses the transport specified by the kubeconfig file.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k describe

Usage:

```
d8 k describe (-f FILENAME | TYPE [NAME_PREFIX | -l label] | TYPE/NAME) [options]
```

Show details of a specific resource or group of resources

Show more...

Show details of a specific resource or group of resources.

Print a detailed description of the selected resources, including related resources such as events or controllers. You may select a single object by name, all objects of that type, provide a name prefix, or label selector. For example:

$ kubectl describe TYPE NAME_PREFIX

will first check for an exact match on TYPE and NAME_PREFIX. If no such resource exists, it will output details for every resource that has a name prefixed with NAME_PREFIX.

Use "kubectl api-resources" for a complete list of supported resources.

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable.
- `--filename`, `-f`

  Filename, directory, or URL to files containing the resource to describe
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-events`

  If true, display events related to the described object.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k diff

Usage:

```
d8 k diff -f FILENAME [options]
```

Diff the live version against a would-be applied version

Show more...

Diff configurations specified by file name or stdin between the current online configuration, and the configuration as it would be if applied.

The output is always YAML.

KUBECTL_EXTERNAL_DIFF environment variable can be used to select your own diff command. Users can use external commands with params too, example: KUBECTL_EXTERNAL_DIFF="colordiff -N -u"

By default, the "diff" command available in your path will be run with the "-u" (unified diff) and "-N" (treat absent files as empty) options.

Exit status: 0 No differences were found. 1 Differences were found. >1 Kubectl or diff failed with an error.

Note: KUBECTL_EXTERNAL_DIFF, if used, is expected to follow that convention.

Options

- `--concurrency`

  Number of objects to process in parallel when diffing against the live version. Larger number = faster, but more memory, I/O and CPU over that shorter period of time.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files contains the configuration to diff

- `--force-conflicts`

  If true, server-side apply will force the changes against conflicts.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--prune`

  Include resources that would be deleted by pruning. Can be used with -l and default shows all resources would be pruned
- `--prune-allowlist`

  Overwrite the default allowlist with for --prune
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-side`

  If true, apply runs in the server instead of the client.
- `--show-managed-fields`

  If true, include managed fields in the diff.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k drain

Usage:

```
d8 k drain NODE [options]
```

Drain node in preparation for maintenance

Show more...

Drain node in preparation for maintenance.

The given node will be marked unschedulable to prevent new pods from arriving. 'drain' evicts the pods if the API server supports https://kubernetes.io/docs/concepts/workloads/pods/disruptions/ eviction https://kubernetes.io/docs/concepts/workloads/pods/disruptions/ . Otherwise, it will use normal DELETE to delete the pods. The 'drain' evicts or deletes all pods except mirror pods (which cannot be deleted through the API server). If there are daemon set-managed pods, drain will not proceed without --ignore-daemonsets, and regardless it will not delete any daemon set-managed pods, because those pods would be immediately replaced by the daemon set controller, which ignores unschedulable markings. If there are any pods that are neither mirror pods nor managed by a replication controller, replica set, daemon set, stateful set, or job, then drain will not delete any pods unless you use --force. --force will also allow deletion to proceed if the managing resource of one or more pods is missing.

'drain' waits for graceful termination. You should not operate on the machine until the command completes.

When you are ready to put the node back into service, use kubectl uncordon, which will make the node schedulable again.

https://kubernetes.io/images/docs/kubectl_drain.svg
Workflowhttps://kubernetes.io/images/docs/kubectl_drain.svg

Options

- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable.
- `--delete-emptydir-data`

  Continue even if there are pods using emptyDir (local data that will be deleted when the node is drained).
- `--disable-eviction`

Force drain to use delete, even if eviction is supported. This will bypass checking PodDisruptionBudgets, use with caution.

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--force`

  Continue even if there are pods that do not declare a controller.
- `--grace-period`

  Period of time in seconds given to each pod to terminate gracefully. If negative, the default value specified in the pod will be used.
- `--ignore-daemonsets`

  Ignore DaemonSet-managed pods.
- `--pod-selector`

  Label selector to filter pods on the node
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--skip-wait-for-delete-timeout`

  If pod DeletionTimestamp older than N seconds, skip waiting for the pod. Seconds must be greater than 0 to skip.
- `--timeout`

  The length of time to wait before giving up, zero means infinite

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k edit

Usage:

```
d8 k edit (RESOURCE/NAME | -f FILENAME) [options]
```

Edit a resource on the server

Show more...

Edit a resource from the default editor.

The edit command allows you to directly edit any API resource you can retrieve via the command-line tools. It will open the editor defined by your KUBE_EDITOR, or EDITOR environment variables, or fall back to 'vi' for Linux or 'notepad' for Windows. When attempting to open the editor, it will first attempt to use the shell that has been defined in the 'SHELL' environment variable. If this is not defined, the default shell will be used, which is '/bin/bash' for Linux or 'cmd' for Windows.

You can edit multiple objects, although changes are applied one at a time. The command accepts file names as well as command-line arguments, although the files you point to must be previously saved versions of resources.

Editing is done with the API version used to fetch the resource. To edit using a specific API version, fully-qualify the resource, version, and group.

The default format is YAML. To edit in JSON, specify "-o json".

The flag --windows-line-endings can be used to force Windows line endings, otherwise the default for your operating system will be used.

In the event an error occurs while updating, a temporary file will be created on disk that contains your unapplied changes. The most common error when updating a resource is another editor changing the resource on the server. When this occurs, you will have to apply your changes to the newer version of the resource, or update your temporary saved copy to include the latest resource version.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files to use to edit the resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--output-patch`

  Output the patch if the resource is edited.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, edit will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--windows-line-endings`

  Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k events

Usage:

```
d8 k events [(-o|--output=)json|yaml|kyaml|name|go-template|go-template-
file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file] [--for TYPE/NAME] [--watch] [--
types=Normal,Warning] [options]
```

List events

Show more...

Display events.

Prints a table of the most important information about events. You can request events for a namespace, for all namespace, or filtered to only those pertaining to a specified resource.

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable.
- `--for`

  Filter events to only those pertaining to the specified resource.
- `--no-headers`

  When using the default output format, don't print headers.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--types`

  Output only events of given types.
- `--watch`, `-w`

  After listing the requested events, watch for more events.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k exec

Usage:

```
d8 k exec (POD | TYPE/NAME) [-c CONTAINER] [flags] -- COMMAND [args...] [options]
```

Execute a command in a container

Show more...

Execute a command in a container.

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--filename`, `-f`

  to use to exec into the resource
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--quiet`, `-q`

  Only print output from the remote session

- `--stdin`, `-i`

  Pass stdin to the container
- `--tty`, `-t`

  Stdin is a TTY

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k explain

Usage:

```
d8 k explain TYPE [--recursive=FALSE|TRUE] [--api-version=api-version-group] [-o|--output=plaintext|plaintext-openapiv2] [options]
```

Get documentation for a resource

Show more...

Describe fields and structure of various resources.

This command describes the fields associated with each supported API resource. Fields are identified via a simple JSONPath identifier:

<type>.<fieldName>[.<fieldName>]

Information about each field is retrieved from the server in OpenAPI format.

Use "kubectl api-resources" for a complete list of supported resources.

Options

- `--api-version`

  Get different explanations for particular API version (API group/version)
- `--output`, `-o`

  Format in which to render the schema (plaintext, plaintext-openapiv2)
- `--recursive`

  Print the fields of fields (Currently only 1 level deep)

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k expose

Usage:

```
d8 k expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [--target-port=number-or-name] [--
name=name] [--external-ip=external-ip-of-service] [--type=type] [options]
```

Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service

Show more...

Expose a resource as a new Kubernetes service.

Looks up a deployment, service, replica set, replication controller or pod by name and uses the selector for that resource as the selector for a new service on the specified port. A deployment or replica set will be exposed as a

service only if its selector is convertible to a selector that service supports, i.e. when the selector contains only the matchLabels component. Note that if no port is specified via --port and the exposed resource has multiple ports, all will be re-used by the new service. Also if no labels are specified, the new service will re-use the labels from the resource it exposes.

Possible resources include (case insensitive):

pod (po), service (svc), replicationcontroller (rc), deployment (deploy), replicaset (rs)

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cluster-ip`

  ClusterIP to be assigned to the service. Leave empty to auto-allocate, or set to 'None' to create a headless service.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--external-ip`

  Additional external IP address (not managed by Kubernetes) to accept for the service. If this IP is routed to a node, the service can be accessed by this IP in addition to its generated service IP.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to expose a service
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--labels`, `-l`

  Labels to apply to the service created by this call.
- `--load-balancer-ip`

  IP to assign to the LoadBalancer. If empty, an ephemeral IP will be created and used (cloud-provider specific).
- `--name`

  The name for the newly created object.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--override-type`

  The method used to override the generated object: json, merge, or strategic.
- `--overrides`

  An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.
- `--port`

  The port that the service should serve on. Copied from the resource being exposed, if unspecified
- `--protocol`

  The network protocol for the service to be created. Default is 'TCP'.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`

  A label selector to use for this service. Only equality-based selector requirements are supported. If empty (the default) infer the selector from the resource being exposed.

- `--session-affinity`

  If non-empty, set the session affinity for the service to this; legal values: 'None', 'ClientIP'
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--target-port`

  Name or number for the port on the container that the service should direct traffic to. Optional.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

  Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k get

Usage:

```
d8 k get [(-o|--output=)json|yaml|kyaml|name|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file|custom-columns|custom-columns-file|wide] (TYPE[.VERSION][.GROUP] [NAME | -l label] | TYPE[.VERSION][.GROUP]/NAME ...) [flags] [options]
```

Display one or many resources

Show more...

Display one or many resources.

Prints a table of the most important information about the specified resources. You can filter the list using a label selector and the --selector flag. If the desired resource type is namespaced you will only see results in the current namespace if you don't specify any namespace.

By specifying the output as 'template' and providing a Go template as the value of the --template flag, you can filter the attributes of the fetched resources.

Use "kubectl api-resources" for a complete list of supported resources.

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--ignore-not-found`

  If set to true, suppresses NotFound error for specific objects that do not exist. Using this flag with commands that query for collections of resources has no effect when no resources are found.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--label-columns`, `-L`

Accepts a comma separated list of labels that are going to be presented as columns. Names are case-sensitive. You can also use multiple flag options like -L label1 -L label2...

- `--no-headers`

  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file, custom-columns, custom-columns-file, wide). See custom columns [https://kubernetes.io/docs/reference/kubectl/#custom-columns], golang template [http://golang.org/pkg/text/template/#pkg-overview] and jsonpath template [https://kubernetes.io/docs/reference/kubectl/jsonpath/].
- `--output-watch-events`

  Output watch event objects when --watch or --watch-only is used. Existing objects are output as initial ADDED events.
- `--raw`

  Raw URI to request from the server. Uses the transport specified by the kubeconfig file.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--scope`

  Cross-namespace listing scope for users without cluster-wide list/watch RBAC: accessible (everything RBAC grants you), projects, system, or project:. Requires -A/--all-namespaces and Deckhouse platform support on the server side.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-print`

  If true, have the server return the appropriate table output. Supports extension APIs and CRDs.
- `--show-kind`

  If present, list the resource type for the requested object(s).
- `--show-labels`

  When printing, show all labels as the last column (default hide labels column)
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--sort-by`

  If non-empty, sort list types using this field specification. The field specification is expressed as a JSONPath expression (e.g. '{.metadata.name}'). The field in the API resource specified by this JSONPath expression must be an integer or a string.
- `--subresource`

  If specified, gets the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--watch`, `-w`

  After listing/getting the requested object, watch for changes.
- `--watch-only`

  Watch for changes to the requested object(s), without listing/getting first.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k kustomize

Usage:

```
d8 k kustomize DIR [options]
```

Build a kustomization target from a directory or URL

Show more...

Build a set of KRM resources using a 'kustomization.yaml' file. The DIR argument must be a path to a directory containing 'kustomization.yaml', or a git repository URL with a path suffix specifying same with respect to the repository root. If DIR is omitted, '.' is assumed.

Options

- `--as-current-user`

  use the uid and gid of the command executor to run the function in the container
- `--enable-alpha-plugins`

  enable kustomize plugins
- `--enable-helm`

  Enable use of the Helm chart inflator generator.
- `--enable-managedby-label`

  enable adding app.kubernetes.io/managed-by
- `--env`, `-e`

  a list of environment variables to be used by functions
- `--helm-api-versions`

  Kubernetes api versions used by Helm for Capabilities.APIVersions
- `--helm-command`

  helm command (path to executable)
- `--helm-debug`

  Enable debug output from the Helm chart inflator generator.
- `--helm-kube-version`

  Kubernetes version used by Helm for Capabilities.KubeVersion
- `--load-restrictor`

  if set to 'LoadRestrictionsNone', local kustomizations may load files from outside their root. This does, however, break the relocatability of the kustomization.
- `--mount`

  a list of storage options read from the filesystem
- `--network`

  enable network access for functions that declare it
- `--network-name`

  the docker network to run the container in
- `--output`, `-o`

  If specified, write output to this path.
- `--reorder`

  Reorder the resources just before output. Use 'legacy' to apply a legacy reordering (Namespaces first, Webhooks last, etc). Use 'none' to suppress a final reordering.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k label

Usage:

```
d8 k label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-version=version]
[options]
```

Update the labels on a resource

Show more...

Update the labels on a resource.

  * A label key and value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 63 characters each.
  * Optionally, the key can begin with a DNS subdomain prefix and a single '/', like example.com/my-app.
  * If --overwrite is true, then existing labels can be overwritten, otherwise attempting to overwrite a label will result in an error.
  * If --resource-version is specified, then updates will use this resource version, otherwise the existing resource-version will be used.

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update the labels
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the labels for a given resource.
- `--local`

  If true, label will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow labels to be overwritten, otherwise reject label updates that overwrite existing labels.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the labels update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a

namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity

- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k logs

Usage:

```
d8 k logs [-f] [-p] (POD | TYPE/NAME) [-c CONTAINER] [options]
```

Print the logs for a container in a pod

Show more...

Print the logs for a container in a pod or specified resource. If the pod has only one container, the container name is optional.

Options

- `--all-containers`

  Get all containers' logs in the pod(s).
- `--all-pods`

  Get logs from all pod(s). Sets prefix to true.
- `--container`, `-c`

  Print the logs of this container
- `--follow`, `-f`

  Specify if the logs should be streamed.
- `--ignore-errors`

  If watching / following pod logs, allow for any errors that occur to be non-fatal
- `--insecure-skip-tls-verify-backend`

  Skip verifying the identity of the kubelet that logs are requested from. In theory, an attacker could provide invalid log content back. You might want to use this if your kubelet serving certificates have expired.
- `--limit-bytes`

  Maximum bytes of logs to return. Defaults to no limit.
- `--max-log-requests`

  Specify maximum number of concurrent logs to follow when using by a selector. Defaults to 5.
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--prefix`

  Prefix each log line with the log source (pod name and container name)
- `--previous`, `-p`

  If true, print the logs for the previous instance of the container in a pod if it exists.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--since`

  Only return logs newer than a relative duration like 5s, 2m, or 3h. Defaults to all logs. Only one of since-time / since may be used.
- `--since-time`

  Only return logs after a specific date (RFC3339). Defaults to all logs. Only one of since-time / since may be used.
- `--tail`

  Lines of recent log file to display. Defaults to -1 with no selector, showing all log lines otherwise 10, if a selector is provided.
- `--timestamps`

  Include timestamps on each line in the log output

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k options

Usage:

```
d8 k options [options]
```

Print the list of flags inherited by all commands

Show more...

Print the list of flags inherited by all commands

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k patch

Usage:

```
d8 k patch (-f FILENAME | TYPE NAME) [-p PATCH|--patch-file FILE] [options]
```

Update fields of a resource

Show more...

Update fields of a resource using strategic merge patch, a JSON merge patch, or a JSON patch.

JSON and YAML formats are accepted.

Note: Strategic merge patch is not supported for custom resources.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, patch will operate on the content of the file, not the server-side resource.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--patch`, `-p`

  The patch to be applied to the resource JSON file.
- `--patch-file`

  A file containing a patch to be applied to the resource.
- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, patch will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

  The type of patch being provided; one of [json merge strategic]

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace, -n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin

Usage:

```
d8 k plugin [flags] [options]
```

Provides utilities for interacting with plugins

Show more...

Provides utilities for interacting with plugins.

Plugins provide extended functionality that is not part of the major command-line distribution. Please refer to the documentation and examples for more information about how write your own plugins.

The easiest way to discover and install plugins is via the kubernetes sub-project krew: [krew.sigs.k8s.io]. To install krew, visit https://krew.sigs.k8s.io/docs/user-guide/setup/install

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin list

Usage:

```
d8 k plugin [flags] list [options]
```

List all visible plugin executables on a user's PATH

Show more...

List all available plugin files on a user's PATH. To see plugins binary names without the full path use --name-only flag.

Available plugin files are those that are: - executable - anywhere on the user's PATH - begin with "kubectl-"

Options

- `--name-only`

  If true, display only the binary name of each plugin, rather than its full path

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k port-forward

Usage:

```
d8 k port-forward TYPE/NAME [options] [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N]
```

Forward one or more local ports to a pod

Show more...

Forward one or more local ports to a pod.

Use resource type/name such as deployment/mydeployment to select a pod. Resource type defaults to 'pod' if omitted.

If there are multiple pods matching the criteria, a pod will be selected automatically. The forwarding session ends when the selected pod terminates, and a rerun of the command is needed to resume forwarding.

Options

- `--address`

  Addresses to listen on (comma separated). Only accepts IP addresses or localhost as a value. When localhost is supplied, kubectl will try to bind on both 127.0.0.1 and ::1 and will fail if neither of these addresses are available to bind.
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k proxy

Usage:

```
d8 k proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [options]
```

Run a proxy to the Kubernetes API server

Show more...

Creates a proxy server or application-level gateway between localhost and the Kubernetes API server. It also allows serving static content over specified HTTP path. All incoming data enters through one port and gets forwarded to the remote Kubernetes API server port, except for the path matching the static content path.

Options

- `--accept-hosts`

  Regular expression for hosts that the proxy should accept.
- `--accept-paths`

  Regular expression for paths that the proxy should accept.
- `--address`

The IP address on which to serve on.

- `--api-prefix`

  Prefix to serve the proxied API under.
- `--append-server-path`

  If true, enables automatic path appending of the kube context server path to each request.
- `--disable-filter`

  If true, disable request filtering in the proxy. This is dangerous, and can leave you vulnerable to XSRF attacks, when used with an accessible port.
- `--keepalive`

  keepalive specifies the keep-alive period for an active network connection. Set to 0 to disable keepalive.
- `--port`, `-p`

  The port on which to run the proxy. Set to 0 to pick a random port.
- `--reject-methods`

  Regular expression for HTTP methods that the proxy should reject (example --reject-methods='POST,PUT,PATCH').
- `--reject-paths`

  Regular expression for paths that the proxy should reject. Paths specified here will be rejected even accepted by --accept-paths.
- `--unix-socket`, `-u`

  Unix socket on which to run the proxy.
- `--www`, `-w`

  Also serve static files from the given directory under the specified prefix.
- `--www-prefix`, `-P`

  Prefix to serve static files under, if static file directory is specified.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k replace

Usage:

```
d8 k replace -f FILENAME [options]
```

Replace a resource by file name or stdin

Show more...

Replace a resource by file name or stdin.

JSON and YAML formats are accepted. If replacing an existing resource, the complete resource spec must be provided. This can be obtained by

    $ kubectl get TYPE NAME -o yaml

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

  The files that contain the configurations to replace.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--kustomize, -k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--raw`

  Raw URI to PUT to the server. Uses the transport specified by the kubeconfig file.
- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, replace will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout

Usage:

```
d8 k rollout SUBCOMMAND [options]
```

Manage the rollout of a resource

Show more...

Manage the rollout of one or many resources.

Valid resource types include:

* deployments
* daemonsets
* statefulsets

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the

server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout history

Usage:

```
d8 k rollout SUBCOMMAND history (TYPE NAME | TYPE/NAME) [flags] [options]
```

View rollout history

Show more...

View previous rollout revisions and configurations.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--revision`

  See the details, including podTemplate of the revision specified
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout pause

Usage:

```
d8 k rollout SUBCOMMAND pause RESOURCE [options]
```

Mark the provided resource as paused

Show more...

Mark the provided resource as paused.

Paused resources will not be reconciled by a controller. Use "kubectl rollout resume" to resume a paused resource. Currently only deployments support being paused.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout restart

Usage:

```
d8 k rollout SUBCOMMAND restart RESOURCE [options]
```

Restart a resource

Show more...

Restart a resource.

Resource rollout will be restarted.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to

golang and jsonpath output formats.

- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename, -f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout resume

Usage:

```
d8 k rollout SUBCOMMAND resume RESOURCE [options]
```

Resume a paused resource

Show more...

Resume a paused resource.

Paused resources will not be reconciled by a controller. By resuming a resource, we allow it to be reconciled again. Currently only deployments support being resumed.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout status

Usage:

```
d8 k rollout SUBCOMMAND status (TYPE NAME | TYPE/NAME) [flags] [options]
```

Show the status of the rollout

Show more...

Show the status of the rollout.

By default 'rollout status' will watch the status of the latest rollout until it's done. If you don't want to wait for the rollout to finish then you can use --watch=false. Note that if a new rollout starts in-between, then 'rollout status' will continue watching the latest revision. If you want to pin to a specific revision and abort if it is rolled over by another revision, use --revision=N where N is the revision you need to watch for.

Options

- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--revision`

  Pin to a specific revision for showing its status. Defaults to 0 (last revision).
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--timeout`

  The length of time to wait before ending watch, zero means never. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).
- `--watch`, `-w`

  Watch the status of the rollout until it's done.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout undo

Usage:

```
d8 k rollout SUBCOMMAND undo (TYPE NAME | TYPE/NAME) [flags] [options]
```

Undo a previous rollout

Show more...

Roll back to a previous rollout.

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--to-revision`

  The revision to rollback to. Default to 0 (last revision).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use

- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k run

Usage:

```
d8 k run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server|client] [--overrides=inline-json]
[--command] -- [COMMAND] [args...] [options]
```

Run a particular image on the cluster

Show more...

Create and run a particular image in a pod.

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--annotations`

  Annotations to apply to the pod.
- `--attach`

  If true, wait for the Pod to start running, and then attach to the Pod as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true. With '--restart=Never' the exit code of the container process is returned.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--command`

  If true and extra arguments are present, use them as the 'command' field in the container, rather than the 'args' field which is the default.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--env`

  Environment variables to set in the container.
- `--expose`

  If true, create a ClusterIP service associated with the pod. Requires `--port`.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  to use to replace the resource.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--image`

  The image for the container to run.
- `--image-pull-policy`

  The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--labels`, `-l`

  Comma separated labels to apply to the pod. Will override previous values.
- `--leave-stdin-open`

  If the pod is started in interactive mode or with stdin, leave stdin open after the first attach completes. By default, stdin will be closed after the first attach completes.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--override-type`

  The method used to override the generated object: json, merge, or strategic.
- `--overrides`

  An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--port`

  The port that this container exposes.
- `--privileged`

  If true, run the container in privileged mode.
- `--quiet`, `-q`

If true, suppress prompt messages.

- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--restart`

  The restart policy for this Pod. Legal values [Always, OnFailure, Never].
- `--rm`

  If true, delete the pod after it exits. Only valid when attaching to the container, e.g. with '--attach' or with '-i/--stdin'.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--stdin`, `-i`

  Keep stdin open on the container in the pod, even if nothing is attached.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--tty`, `-t`

  Allocate a TTY for the container in the pod.
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections

insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k scale

Usage:

```
d8 k scale [--resource-version=version] [--current-replicas=count] --replicas=COUNT (-f FILENAME | TYPE NAME)
[options]
```

Set a new size for a deployment, replica set, or replication controller

Show more...

Set a new size for a deployment, replica set, replication controller, or stateful set.

Scale also allows users to specify one or more preconditions for the scale action.

If --current-replicas or --resource-version is specified, it is validated before the scale is attempted, and it is guaranteed that the precondition holds true when the scale is sent to the server.

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to

golang and jsonpath output formats.

- `--current-replicas`

  Precondition for current size. Requires that the current size of the resource match this value in order to scale. -1 (default) for no condition.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to set a new size
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--replicas`

  The new desired number of replicas. Required.
- `--resource-version`

  Precondition for resource version. Requires that the current resource version match this value in order to scale.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a scale operation, zero means don't wait. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set

Usage:

```
d8 k set SUBCOMMAND [options]
```

Set specific features on objects

Show more...

Configure application resources.

These commands help you make changes to existing application resources.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server

- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set env

Usage:

```
d8 k set SUBCOMMAND env RESOURCE/NAME KEY_1=VAL_1 ... KEY_N=VAL_N [options]
```

Update environment variables on a pod template

Show more...

Update environment variables on a pod template.

List environment variable definitions in one or more pods, pod templates. Add, update, or remove container environment variable definitions in one or more pod templates (within replication controllers or deployment configurations). View or modify the environment variable definitions on all containers in the specified pods or pod templates, or just those that match a wildcard.

If "--env -" is passed, environment variables can be read from STDIN using the standard env syntax.

Possible resources include (case insensitive):

pod (po), replicationcontroller (rc), deployment (deploy), daemonset (ds), statefulset (sts), cronjob (cj), replicaset (rs)

Options

- `--all`

  If true, select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--containers`, `-c`

  The names of containers in the selected pod templates to change - may use wildcards
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--env`, `-e`

  Specify a key-value pair for an environment variable to set into each container.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files the resource to update the env
- `--from`

  The name of a resource from which to inject environment variables
- `--keys`

  Comma-separated list of keys to import from specified resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the environment and any changes in the standard format. this flag will removed when we have kubectl view env.
- `--local`

  If true, set env will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

If true, allow environment to be overwritten, otherwise reject updates that overwrite existing environment.

- `--prefix`

  Prefix to append to variable names
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resolve`

  If true, show secret or configmap references when listing variables
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server

- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set image

Usage:

```
d8 k set SUBCOMMAND image (-f FILENAME | TYPE NAME) CONTAINER_NAME_1=CONTAINER_IMAGE_1 ...
CONTAINER_NAME_N=CONTAINER_IMAGE_N [options]
```

Update the image of a pod template

Show more...

Update existing container image(s) of resources.

Possible resources include (case insensitive):

pod (po), replicationcontroller (rc), deployment (deploy), daemonset (ds), statefulset (sts), cronjob (cj), replicaset (rs)

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, set image will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile,

jsonpath, jsonpath-as-json, jsonpath-file).

- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server

- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set resources

Usage:

```
d8 k set SUBCOMMAND resources (-f FILENAME | TYPE NAME)  ([--limits=LIMITS & --requests=REQUESTS] [options]
```

Update resource requests/limits on objects with pod templates

Show more...

Specify compute resource requirements (CPU, memory) for any resource that defines a pod template. If a pod is successfully scheduled, it is guaranteed the amount of resource requested, but may burst up to its specified limits.

For each compute resource, if a limit is specified and a request is omitted, the request will default to the limit.

Possible resources include (case insensitive): Use "kubectl api-resources" for a complete list of supported resources..

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--containers`, `-c`

  The names of containers in the selected pod templates to change, all containers are selected by default - may use wildcards
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.

- `--limits`

  The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--local`

  If true, set resources will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--requests`

  The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set selector

Usage:

```
d8 k set SUBCOMMAND selector (-f FILENAME | TYPE NAME) EXPRESSIONS [--resource-version=version] [options]
```

Set the selector on a resource

Show more...

Set the selector on a resource. Note that the new selector will overwrite the old selector if the resource had one prior to the invocation of 'set selector'.

A selector must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 63 characters. If --resource-version is specified, then updates will use this resource version, otherwise the existing resource-version will be used. Note: currently selectors can only be set on Service objects.

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

  identifying the resource.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output, -o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the selectors update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set serviceaccount

Alias: `sa`

Usage:

```
d8 k set SUBCOMMAND serviceaccount (-f FILENAME | TYPE NAME) SERVICE_ACCOUNT [options]
```

Update the service account of a resource

Show more...

Update the service account of pod template resources.

Possible resources (case insensitive) can be:

replicationcontroller (rc), deployment (deploy), daemonset (ds), job, replicaset (rs), statefulset

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

Filename, directory, or URL to files identifying the resource to get from a server.

- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, set serviceaccount will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version

- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set subject

Usage:

```
d8 k set SUBCOMMAND subject (-f FILENAME | TYPE NAME) [--user=username] [--group=groupname] [--
serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Update the user, group, or service account in a role binding or cluster role binding

Show more...

Update the user, group, or service account in a role binding or cluster role binding.

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files the resource to update the subjects
- `--group`

  Groups to bind to the role
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

If true, set subject will NOT contact api-server but run locally.

- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--serviceaccount`

  Service accounts to bind to the role
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the role

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k taint

Usage:

```
d8 k taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]
```

Update the taints on one or more nodes

Show more...

Update the taints on one or more nodes.

  * A taint consists of a key, value, and effect. As an argument here, it is expressed as key=value:effect.
  * The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 253 characters.
  * Optionally, the key can begin with a DNS subdomain prefix and a single '/', like example.com/my-app.
  * The value is optional. If given, it must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores, up to 63 characters.
  * The effect must be NoSchedule, PreferNoSchedule or NoExecute.
  * Currently taint can only apply to node.

Options

- `--all`

  Select all nodes in the cluster
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow taints to be overwritten, otherwise reject taint updates that overwrite existing taints.

- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top

Usage:

```
d8 k top [options]
```

Display resource (CPU/memory) usage

Show more...

Display resource (CPU/memory) usage.

This command provides a view of recent resource consumption for nodes and pods. It fetches metrics from the Metrics Server, which aggregates this data from the kubelet on each node. The Metrics Server must be installed and running in the cluster for this command to work.

The metrics shown are specifically optimized for Kubernetes autoscaling decisions, such as those made by the Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA). Because of this, the values may not match those from standard OS tools like 'top', as the metrics are designed to provide a stable signal for autoscalers rather than for pinpoint accuracy.

When to use this command:

* For on-the-fly spot-checks of resource usage (e.g. identify which pods are consuming the most resources at a glance, or get a quick sense of the load on your nodes)
* Understand current resource consumption patterns
* Validate the behavior of your HPA or VPA configurations by seeing the metrics they use for scaling decisions.

It is not intended to be a replacement for full-featured monitoring solutions. Its primary design goal is to provide a low-overhead signal for autoscalers, not to be a perfectly accurate monitoring tool. For high-accuracy reporting, historical analysis, dashboarding, or alerting, you should use a dedicated monitoring solution.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top node

Aliases: `nodes`, `no`

Usage:

```
d8 k top node [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of nodes

Show more...

Display resource (CPU/memory) usage of nodes.

The top-node command allows you to see the resource consumption of nodes.

Options

- `--no-headers`
  If present, print output without headers
- `--selector`, `-l`
  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-capacity`
  Print node resources based on Capacity instead of Allocatable(default) of the nodes.
- `--show-swap`
  Print node resources related to swap memory.
- `--sort-by`
  If non-empty, sort nodes list using specified field. The field can be either 'cpu' or 'memory'.
- `--use-protocol-buffers`
  Enables using protocol-buffers to access Metrics API.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--as-user-extra`
  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top pod

Aliases: `pods`, `po`

Usage:

```
d8 k top pod [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of pods

Show more...

Display resource (CPU/memory) usage of pods.

The 'top pod' command allows you to see the resource consumption of pods.

Due to the metrics pipeline delay, they may be unavailable for a few minutes since pod creation.

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--containers`

  If present, print usage of containers within a pod.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.

- `--no-headers`

  If present, print output without headers.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-swap`

  Print pod resources related to swap memory.
- `--sort-by`

  If non-empty, sort pods list using specified field. The field can be either 'cpu' or 'memory'.
- `--sum`

  Print the sum of the resource usage
- `--use-protocol-buffers`

  Enables using protocol-buffers to access Metrics API.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k uncordon

Usage:

```
d8 k uncordon NODE [options]
```

Mark node as schedulable

Show more...

Mark node as schedulable.

Options

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k version

Usage:

```
d8 k version [options]
```

Print the client and server version information

Show more...

Print the client and server version information for the current context.

Options

- `--client`

  If true, shows client version only (no server required).
- `--output`, `-o`

  One of 'yaml' or 'json'.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k wait

Usage:

```
d8 k wait ([-f FILENAME] | resource.group/resource.name | resource.group [(-l label | --all)]) [--for=create|--
for=delete|--for condition=available|--for=jsonpath='{}'[=value]] [options]
```

Wait for a specific condition on one or many resources

Show more...

Wait for a specific condition on one or many resources.

The command takes multiple resources and waits until the specified condition is seen in the Status field of every given resource.

Alternatively, the command can wait for the given set of resources to be created or deleted by providing the "create" or "delete" keyword as the value to the --for flag.

A successful message will be printed to stdout indicating when the specified condition has been met. You can use -o option to change to output destination.

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  identifying the resource.
- `--for`

  The condition to wait on: [create|delete|condition=condition-name[=condition-value]|jsonpath='{JSONPath expression}'=[JSONPath value]]. The default condition-value is true. Condition values are compared after Unicode simple case folding, which is a more general form of case-insensitivity.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, kyaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2)

- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up. Zero means check once and don't wait, negative means wait for a week.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--as-user-extra`

  User extras to impersonate for the operation, this flag can be repeated to specify multiple values for the same key.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex|trace)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 mirror

Copy Deckhouse Kubernetes Platform distribution to the local filesystem or third-party registry

Show more...

Copy Deckhouse Kubernetes Platform distribution to the local filesystem or the air-gapped registries.

For more information on how to use it, consult the docs at https://deckhouse.io/products/kubernetes-platform/documentation/latest/installing/#manual-loading-of-dkp-images-and-vulnerability-db-into-a-private-registry

LICENSE NOTE: The d8 mirror functionality is exclusively available to users holding a valid license for any commercial version of the Deckhouse Kubernetes Platform.

© Flant JSC 2025

d8 mirror pull

Usage:

```
d8 mirror pull <images-bundle-path> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the local filesystem

Show more...

Download Deckhouse Kubernetes Platform distribution to the local filesystem.

This command downloads the Deckhouse Kubernetes Platform distribution bundle
containing specific platform releases and it's modules,
to be pushed into the air-gapped container registry at a later time.

For more information on how to use it, consult the docs at
https://deckhouse.io/products/kubernetes-platform/documentation/latest/installing/#manual-loading-of-dkp-images-and-vulnerability-db-into-a-private-registry

Additional configuration options for the d8 mirror family of commands are available as environment variables:

 * $SSL_CERT_FILE — Path to the SSL certificate. If the variable is set, system certificates are not used;
 * $SSL_CERT_DIR — List of directories to search for SSL certificate files, separated by a colon.
If set, system certificates are not used. More info at
https://docs.openssl.org/1.0.2/man1/c_rehash/;
 * $HTTP_PROXY/$HTTPS_PROXY — URL of the proxy server for HTTP(S) requests to hosts that are not listed in the $NO_PROXY;
 * $NO_PROXY — Comma-separated list of hosts to exclude from proxying.
Supported value formats include IP's', CIDR notations (1.2.3.4/8), domains, and asterisk sign (*).
The IP addresses and domain names can also include a literal port number (1.2.3.4:80).
The domain name matches that name and all the subdomains.
The domain name with a leading . matches subdomains only.
For example, foo.com matches foo.com and bar.foo.com; .y.com matches x.y.com but does not match y.com.
A single asterisk * indicates that no proxying should be done;

LICENSE NOTE:
The d8 mirror functionality is exclusively available to users holding a
valid license for any commercial version of the Deckhouse Kubernetes Platform.

© Flant JSC 2025

Options

- `--deckhouse-cli-tag`

  Specific d8 CLI version to mirror, e.g. "v0.13.1". Without this flag the newest published stable version is mirrored.

  The CLI binary lives at the deckhouse-cli registry segment, next to the plugins catalog; a registry that publishes no such repository is reported in the summary instead of failing the pull, unless this flag asked for a version.
- `--deckhouse-tag`

  Specific Deckhouse build tag to pull. Conflicts with --since-version and --include-platform. If registry contains release channel image for specified tag, all release channels in the bundle will be pointed to it.
- `--dry-run`

  Print what would be pulled without downloading any images. Useful for fast validation of flags and filters.
- `--exclude-module`, `-e`

  Blacklist specific modules from downloading. Format is "module-name[@constraint]", the same dialect as --include-module, quoting included. Use one flag per each module. Overridden by use of --include-module.
- `--exclude-package`

  Blacklist specific packages from downloading. Format is "package-name[@constraint]", the same dialect as --include-module, quoting included. Use one flag per each package. Overridden by use of --include-package.
- `--force`

  Overwrite existing bundle packages if they are conflicting with current pull operation.
- `--gost-digest`

  Calculate GOST R 34.11-2012 STREEBOG digest for downloaded bundle
- `--ignore-suspend`

  Ignore suspended release channels and continue mirroring. Use with caution.
- `--images-bundle-chunk-size`, `-c`

  Split resulting bundle file into chunks of at most N gigabytes
- `--include-module`, `-i`

  Whitelist specific modules for downloading. Format is "module-name[@constraint]". Use one flag per each module. Disables blacklisting by --exclude-module.

  Quote the whole value when the constraint uses >= or <=: --include-module "module-name@>=1.3.0". Unquoted, the shell takes > as a redirection and d8 receives a module with no version.

  A constraint keeps the latest patch in each minor it covers. Versions named with >= or <= are kept as well. An exact tag (=) pins one tag and publishes it to the release channels.

  All other constraints also pull the versions the release channels point at. These are added on top of the range without being checked against it, so the upper bound does not apply to them: "module-name@>=1.3.0 <=1.4.0" still pulls v1.4.1 if a release channel points at it. The examples below leave them out.

  Given v1.0.0, v1.1.0, v1.2.0, v1.3.0, v1.3.3, v1.4.0, v1.4.1:
  module-name only what the release channels point at, like a pull with no filters
  module-name@1.3.0 >=1.3.0 <2.0.0, same major line: v1.3.3, v1.4.1
  module-name@~1.3.0 >=1.3.0 <1.4.0: v1.3.3
  module-name@^1.3.0 >=1.3.0 <2.0.0: v1.3.3, v1.4.1
  "module-name@>=1.3.0" the same, plus the named v1.3.0
  "module-name@>=1.3.0 <=1.4.0" v1.3.0, v1.3.3, v1.4.0
  module-name@=v1.3.0 only v1.3.0, published to every release channel
  module-name@=v1.3.0+stable only v1.3.0, published to stable
  module-name@=bobV1 only the bobV1 tag, published to every release channel

  For a 0.x module the bare form spans the whole 0.x line (0.4.0 means >=0.4.0 <1.0.0) while the caret locks

the minor (^0.4.0 means >=0.4.0 <0.5.0).

- `--include-package`

  Whitelist specific packages for downloading. Format is "package-name[@constraint]", the same dialect as --include-module, quoting included. Use one flag per each package. Disables blacklisting by --exclude-package.

  Packages live under the packages/ registry segment, with release metadata under packages//version.
- `--include-platform`

  Select platform releases to download by a semver constraint. Same dialect as the version part of --include-module.
  Conflicts with --since-version and --deckhouse-tag.

  Always quote the value: > and < are shell redirections.

  A constraint keeps the latest patch in each minor it covers. Versions named with >= or <= are kept as well. An exact tag (=) pins one release and propagates it to the release channels, just like --deckhouse-tag.

  Given platform releases v1.63.x .. v1.71.x:
  ">=1.64 <=1.68" latest patch per minor in v1.64..v1.68, plus v1.64.0 and v1.68.0
  "~1.65.0" latest v1.65.x patch
  "^1.65.0" latest patch per minor from v1.65.x up
  "1.65.0" same as ^1.65.0 here: platform majors are always >= 1
  "=v1.65.3" only v1.65.3, published to every release channel
  "=v1.65.3+stable" only v1.65.3; the +channel suffix is accepted but changes nothing here
- `--include-plugin`

  Mirror a specific d8 CLI plugin in addition to the automatic selection. Format is "plugin-name[@constraint]", the same dialect as --include-module, quoting included. Use one flag per each plugin.

  Plugins live under the deckhouse-cli/plugins registry segment. Without this flag, plugins required by the mirrored modules (and their plugin dependencies) are selected automatically.
- `--insecure`

  Interact with registries over HTTP.
- `--installer-tag`

  Specific Deckhouse installer build tag to pull. If not specified, the latest tag for the installer will be pulled from the registry path.
- `--license, -l`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=<>.
- `--modules-path-suffix`

  Registry path suffix to read modules from, relative to the source repo. Use "/" to read modules from the repo root.
- `--no-installer`

  Do not pull Deckhouse installer into bundle.
- `--no-modules`

  Do not pull Deckhouse modules into bundle.
- `--no-packages`

  Do not pull Deckhouse packages into bundle.
- `--no-platform`

  Do not pull Deckhouse Kubernetes Platform into bundle.
- `--no-pull-resume`

  Do not continue last unfinished pull operation and start from scratch.
- `--no-security-db`

  Do not pull security databases into bundle.
- `--only-extra-images`

  Pull only extra images for modules (additional images like security databases, scanners, etc.) without pulling main module images.
- `--proxy-registry`

  Pull from a proxy/caching registry that does not implement the registry catalog API.

  Instead of calling the registry's "list tags" endpoint (which proxy registries typically return empty), this mode

probes individual tags by incrementing patch -> minor -> major from the version explicitly named via --include-platform / --include-module. The probe stops once both a new patch and a new minor of the current major fail to resolve, then attempts the next major; if that also fails the probe terminates and downloads what was discovered.

Requires --include-platform when platform is not skipped via --no-platform, and at least one --include-module when modules are not skipped via --no-modules. --exclude-module and --no-platform are respected.

Cannot be combined with --deckhouse-tag or --since-version (use --include-platform's lower bound instead).

- `--since-version`

  Minimal Deckhouse release to pull. Ignored if above current Rock Solid release. Conflicts with --deckhouse-tag.
- `--skip-vex-images`

  Do not pull VEX images.
- `--source`

  Source registry to pull Deckhouse images from (format: registry-host[:port]/path).
- `--source-login`

  Source registry login.
- `--source-password`

  Source registry password.
- `--tls-skip-verify`

  Disable TLS certificate validation.
- `--tmp-dir`

  Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;
- `--verbose-summary`

  List every module and package in the end-of-pull summary with its resolved versions (and VEX count, when present), instead of just the totals. Output only - it does not change what is pulled.

d8 mirror push

Usage:

```
d8 mirror push [images-bundle-path] <registry> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the third-party registry

Show more...

Upload Deckhouse Kubernetes Platform distribution bundle to the third-party registry.

This command pushes the Deckhouse Kubernetes Platform distribution into the specified container registry.

When the registry path ends with an edition (ce, be, se, se-plus, ee, fe, cse), e.g. registry.example.com/deckhouse/ee, the edition-independent installer and d8 CLI (with its plugins) are pushed one level up, outside the edition: registry.example.com/deckhouse/installer and registry.example.com/deckhouse/deckhouse-cli. That is where the Deckhouse registry keeps them, once for all editions. Everything else is pushed under the given path.

For more information on how to use it, consult the docs at https://deckhouse.io/products/kubernetes-platform/documentation/latest/installing/#manual-loading-of-dkp-images-and-vulnerability-db-into-a-private-registry

Additional configuration options for the d8 mirror family of commands are available as environment variables:

 * $SSL_CERT_FILE — Path to the SSL certificate. If the variable is set, system certificates are not used;
 * $SSL_CERT_DIR — List of directories to search for SSL certificate files, separated by a colon. If set, system certificates are not used. More info at https://docs.openssl.org/1.0.2/man1/c_rehash/;
 * $HTTP_PROXY/$HTTPS_PROXY — URL of the proxy server for HTTP(S) requests to hosts that are not listed in the $NO_PROXY;
 * $NO_PROXY — Comma-separated list of hosts to exclude from proxying.
Supported value formats include IP's', CIDR notations (1.2.3.4/8), domains, and asterisk sign (*).
The IP addresses and domain names can also include a literal port number (1.2.3.4:80).
The domain name matches that name and all the subdomains.
The domain name with a leading . matches subdomains only.
For example, foo.com matches foo.com and bar.foo.com; .y.com matches x.y.com but does not match y.com.

A single asterisk * indicates that no proxying should be done;

LICENSE NOTE:
The d8 mirror functionality is exclusively available to users holding a
valid license for any commercial version of the Deckhouse Kubernetes Platform.

© Flant JSC 2025

Options

- `--file`

  `Path` to a tar or chunked package to push. May be repeated to push multiple packages. Can be used instead of the bundle directory argument or combined with it.
- `--insecure`

  Interact with registries over HTTP.
- `--modules-path-suffix`

  Registry path suffix to push modules to, relative to the target repo. Use "/" to push modules to the repo root.
- `--registry-login`, `-u`

  Username to log into the target registry.
- `--registry-password`, `-p`

  Password to log into the target registry.
- `--tls-skip-verify`

  Disable TLS certificate validation.
- `--tmp-dir`

  Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;

d8 network

Alias: `n`

A group of commands to operate network related tasks in The Deckhouse Ecosystem.

Show more...

A group of commands to operate network related tasks in The Deckhouse Ecosystem.

© Flant JSC 2025

d8 network cni-migration

Usage:

```
d8 network cni-migration [options]
```

A group of commands to switch CNI in the cluster

Show more...

A group of commands to switch the CNI (Container Network Interface) provider in the DKP.

The migration process is handled automatically by an in-cluster controller.
This CLI tool is used to trigger the migration and monitor its status.

Workflow:

1. 'd8 network cni-migration switch --to-cni <CNI>' - Initiates the migration.
   This creates a CNIMigration resource, which triggers the deployment of the migration agent.
   The agent then performs all necessary steps (validation, node checks, CNI switching).

2. 'd8 network cni-migration watch' - (Optional) Monitors the progress of the migration.
   Since the process is automated, this command simply watches the status.

3. 'd8 network cni-migration cleanup' - Cleans up the migration resources after completion.

d8 network cni-migration cleanup

Usage:

```
d8 network cni-migration cleanup [options]
```

Cleans up resources created during CNI switching

d8 network cni-migration switch

Usage:

```
d8 network cni-migration switch [options]
```

Initiates the CNI switching

Options

- `--to-cni`

  Target CNI provider to switch to. Supported values: cilium, flannel, simple-bridge

d8 network cni-migration watch

Usage:

```
d8 network cni-migration watch [options]
```

Monitors the CNI switching progress

d8 network istio

Usage:

```
d8 network istio [options]
```

Run an interactive istioctl debug container

Show more...

Start an interactive debug container with istioctl and the RBAC needed to inspect pods in a target namespace (get/list pods and create pods/portforward).

The ServiceAccount is created in --namespace. Roles and RoleBindings are created in --target-namespace (workload pods) and --istio-namespace (default d8-istio): pods get/list, pods/portforward create, and in the Istio namespace also serviceaccounts get plus serviceaccounts/token create for istiod RPC auth (istioctl proxy-status). No Istio config mutations. The debug image is taken from ConfigMap d8-system/debug-container unless --image is set.

The pod is deleted when the session ends. RBAC objects are left in place so the next run can reuse them.

© Flant JSC 2026

Options

- `--context`

  The name of the kubeconfig context to use
- `--image`

  Debug container image (default: ConfigMap d8-system/debug-container)
- `--istio-namespace`

  Istio control-plane namespace for read-only access (istioctl proxy-status)
- `--kubeconfig`, `-k`

  Path to kubeconfig file
- `--namespace`, `-n`

  Namespace for the ServiceAccount and debug pod (default: current kubeconfig namespace)
- `--target-namespace`

  Namespace whose workload pods istioctl should inspect (default: --namespace)

d8 package

Package build and bootstrap tool for containerized packages

Show more...

A plugin for building and bootstrapping packages.

This plugin helps you:
• Bootstrap new packages from templates with registry configuration
• Build containerized packages and push them to container registries
• Manage package lifecycle with semantic versioning

d8 package bootstrap

Usage:

```
d8 package bootstrap [options]
```

Bootstrap a new package (module or application)

Show more...

Bootstrap a new package from templates.

Use 'bootstrap module <name>' for a module (with hooks, CRDs).
Use 'bootstrap app <name>' for an application package (per-instance resources).

d8 package bootstrap app

Alias: `application`

Usage:

```
d8 package bootstrap app <name> [options]
```

Bootstrap a new application package

Show more...

Bootstrap a new application package from templates.

- Creates a package with type Application in package.yaml
- Includes hooks
- Initializes a git repository with an initial commit

Options

- `--extended`

  Generate extended files in the package
- `--hooks`

  Generate hooks in the package
- `--output`, `-o`

  Custom path for the package (default: /)
- `--werf`

  Use werf file for images

d8 package bootstrap module

Usage:

```
d8 package bootstrap module <name> [options]
```

Bootstrap a new module package

Show more...

Bootstrap a new module package from templates.

- Creates a package with type Module in package.yaml
- Includes hooks and CRDs; templates use package name
- Initializes a git repository with an initial commit

Options

- `--extended`

  Generate extended files in the package
- `--hooks`

  Generate hooks in the package
- `--output`, `-o`

  Custom path for the package (default: /)
- `--werf`

  Use werf file for images

d8 package build

Usage:

```
d8 package build --version <version> [--repository <repository>] [--force] [options]
```

Build and push a package to a container registry

Show more...

Build a package using Werf and push it to a container registry.

This command:
- Validates the package structure (package.yaml)
- Builds container images using Werf in a Docker container
- Pushes the built images to the specified registry
- Tags images with semantic version

Environment Variables:
```
PACKAGE_BUILD_REPOSITORY              Registry URL for build command
PACKAGE_BUILD_REPOSITORY_USER         Registry username for authentication
PACKAGE_BUILD_REPOSITORY_TOKEN        Registry token for authentication
PACKAGE_BUILD_FINAL_REPOSITORY        Final registry URL for the published artifact
PACKAGE_BUILD_FINAL_REPOSITORY_USER   Final registry username for authentication
PACKAGE_BUILD_FINAL_REPOSITORY_TOKEN  Final registry token for authentication
PACKAGE_BUILD_INSECURE                Allow plain HTTP and skip TLS verification
```

Options

- `--debug`

  enable debug logging
- `--final-repo`

  Final repository (env: PACKAGE_BUILD_FINAL_REPOSITORY)
- `--final-token`

  Final registry token (env: PACKAGE_BUILD_FINAL_REPOSITORY_TOKEN)
- `--final-user`

  Final registry user (env: PACKAGE_BUILD_FINAL_REPOSITORY_USER)
- `--force`, `-f`

  force update version in registry
- `--insecure`

  Allow plain HTTP and skip TLS verification for every registry used by the build, including the final repository and base-image pulls (env: PACKAGE_BUILD_INSECURE)
- `--repo`, `-r`

  Repository (env: PACKAGE_BUILD_REPOSITORY)
- `--sign`

  sign package with certificate and key. --sign-cert and --sign-key are required
- `--sign-cert`

  sign certificate path or base64 string (env: PACKAGE_BUILD_SIGN_CERT)
- `--sign-key`

  sign key path or base64 string or vault url (env: PACKAGE_BUILD_SIGN_KEY)
- `--token`, `-t`

  Registry token (env: PACKAGE_BUILD_REPOSITORY_TOKEN)
- `--user`, `-u`

  Registry user (env: PACKAGE_BUILD_REPOSITORY_USER)
- `--version`, `-v`

  Package version

d8 package doc

Usage:

```
d8 package doc [linter] [options]
```

Show documentation for verify linters and rules

Show more...

Show documentation for the linters run by 'package verify'.

Use 'package doc' to list every linter.

Use 'package doc <linter>' to see what a linter checks, the rules it runs with
their default severities, and the .pkglint.yaml keys that configure it.

Use 'package doc <linter> --rule <rule>' to see what makes a single rule report a finding and how to satisfy it.

A rule's impact caps the severity of its findings: ignored hides them, warn reports them without failing, error fails verification.

Options

- `--rule`

  Show documentation for a single rule of the linter

d8 package render

Usage:

```
d8 package render [options]
```

Render package templates to Kubernetes manifests

Show more...

Render a package's templates to Kubernetes manifests using stubbed runtime values.

Use 'package render' in a package directory to preview the manifests that would be installed in a cluster. Image digests, registry and credentials are stubbed, so the output is a smoke-check preview, not a faithful representation of a real cluster.

Use 'package render --remote <registry-path>/<package>:<version>' to render a published bundle image instead of the current directory. The image must exist and must carry an explicit tag or digest. Only the bundle image is renderable: the release image ships without a templates directory.

Registry credentials default to the ambient Docker keychain. Pass --remote-user and --remote-password to authenticate explicitly, or set them in the environment:

```
PACKAGE_REMOTE_USER      fallback for --remote-user
PACKAGE_REMOTE_PASSWORD  fallback for --remote-password
```

Use --file to print only the manifests rendered from the template with that file name, and --render-file to write a clean render (without '# Source:' headers) to a file.

Options

- `--file`

  Show only manifests rendered from the template with this file name
- `--remote`, `-r`

  Render a published bundle image by its reference, /:
- `--remote-password`

  Registry password or token for --remote (env: PACKAGE_REMOTE_PASSWORD)
- `--remote-user`

  Registry user for --remote (env: PACKAGE_REMOTE_USER)
- `--render-file`

  Write the clean render (without '# Source:' headers) to this file path

d8 package verify

Usage:

```
d8 package verify [options]
```

Verify a package structure and configuration

Show more...

Verify a package structure and configuration.

Use 'package verify' to verify the current package directory against the 'static' linter settings.

Use 'package verify remote <registry-path> <package>' to verify a published package instead.

Linter configuration can be customized via .pkglint.yaml in the package root.

Options

- `--hide-warnings`

  Hide warning-level messages
- `--lint-config`

  Path to lint config file
- `--show-ignored`

  Show ignored-level messages

d8 package verify remote

Usage:

```
d8 package verify remote <registry-path> <package> [options]
```

Verify a published package from a registry

Show more...

Verify a published package.

The arguments are the registry path the package is published under and the package name. Build publishes two images per version, both tagged with that version:

```
  bundle   <registry-path>/<package>:<version>
         checked against the 'remote.bundle' linter settings
  release  <registry-path>/<package>/version:<version>
         checked against the 'remote.release' linter settings
```

Without --version the latest published version is verified, chosen by comparing the repository tags as semver. The bundle image carries the package itself and is always verified; add --release to verify the release image alongside it, in which case both are verified concurrently and each must exist.

Lint settings are read from the working directory, not from the image: published images do not carry a .pkglint.yaml.

Options

- `--release`

  Also verify the release image published for the version
- `--version`

  Version to verify (default: the latest published version)

Global options

- `--hide-warnings`

  Hide warning-level messages
- `--lint-config`

  Path to lint config file
- `--show-ignored`

  Show ignored-level messages

d8 snapshot

Snapshot operations (create, delete, download, restore, upload, get)

d8 snapshot create

Usage:

```
d8 snapshot create <name> [options]
```

Create a Snapshot of a namespace

Show more...

Create a Snapshot object that captures a namespace's state/configuration.

The Snapshot is created in the target namespace (defaults to the kubeconfig context namespace). An empty selector captures the whole namespace; pass --selector to capture only the objects matching the given labels (sets spec.resourceSelector.matchLabels).

The Snapshot is reconciled asynchronously by the state-snapshotter controller. Use --wait

to block until it reports Ready, after which it can be listed, downloaded, and imported.

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--namespace`, `-n`

  namespace to create the Snapshot in (defaults to the kubeconfig context namespace)
- `--output`, `-o`

  Output format: name|json|yaml
- `--selector`, `-l`

  capture only objects matching this label selector (e.g. app=demo,tier=db); sets spec.resourceSelector.matchLabels
- `--timeout`

  timeout for --wait
- `--wait`

  wait until the Snapshot reports Ready

d8 snapshot delete

Usage:

```
d8 snapshot delete [name...] [options]
```

Delete one or more Snapshots

Show more...

Delete Snapshot objects in a namespace.

Targets are selected in exactly one of three ways:
 - by name: d8 snapshot delete snap-a snap-b -n demo
 - by selector: d8 snapshot delete -l app=demo -n demo
 - all: d8 snapshot delete --all -n demo

Deleting a Snapshot lets the state-snapshotter controller garbage-collect the associated SnapshotContent and data artifacts. Use --wait to block until the Snapshot objects are fully removed from the API server.

Options

- `--all`

  delete all Snapshots in the namespace
- `--context`

  The name of the kubeconfig context to use
- `--ignore-not-found`

  do not error if a named Snapshot does not exist
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--namespace`, `-n`

  namespace of the Snapshots (defaults to the kubeconfig context namespace)
- `--selector`, `-l`

  delete Snapshots matching this label selector (e.g. app=demo,tier=db)
- `--timeout`

  timeout for --wait
- `--wait`

  wait until the Snapshots are fully deleted

d8 snapshot describe

Usage:

```
d8 snapshot describe [flags] <name> [options]
```

Show the snapshot hierarchy as a tree

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--namespace`, `-n`

  snapshot namespace (defaults to the kubeconfig context namespace)

d8 snapshot download

Usage:

```
d8 snapshot download [flags] <snapshot> [options]
```

Download a snapshot to a local directory tree

Show more...

Download a Snapshot CR's manifest tree and volume data into a local directory tree
(consumed by 'd8 snapshot upload' or 'd8 snapshot restore').

--publish selects how each data leaf's volume bytes are streamed from its DataExport
exporter pod. With --publish=false (or when autodetection picks it), bytes come straight
from the exporter's in-cluster service, trusting only its internal CA (status.ca). With
--publish=true, bytes come through the storage-foundation-published Ingress endpoint
(status.publicURL) instead, so a kubeconfig without direct network access to the cluster's
internal service network can still download. If --publish is not given, the command probes
whether the in-cluster exporter endpoint is reachable and picks accordingly. Reusing an
existing DataExport upgrades its spec.publish from false to true when --publish=true is
requested, but never downgrades it back to false, so a concurrent run streaming through the
public endpoint is never torn down. IMPORTANT: the publish path works only with a kubeconfig
authenticated by a bearer token. Ingress terminates TLS with its own certificate and does not
forward the client's TLS certificate to the exporter pod, so a certificate-based kubeconfig
receives a 401 when --publish=true.

Options

- `--cleanup`

  delete the per-volume DataExport (and its server-side export chain) after each volume completes; --cleanup=false leaves them in the cluster for debugging
- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--max-parallel-downloads`

  global cap on concurrent whole-volume-stream downloads across all nodes (independent of --workers and --per-volume-concurrency)
- `--namespace`, `-n`

  snapshot namespace (required)
- `--node`

  restrict download to a single node subtree; format '/' (e.g. --node DemoVirtualDisk/bk-disk-a, --node Snapshot/my-snap); the generated snapshot CR name form (e.g. DemoVirtualDiskSnapshot/nss-child-abc) is still accepted
- `--output`, `-o`

  root output directory (required)
- `--per-volume-concurrency`

  maximum parallel chunk/file downloads per volume; lower to 1 on a distant or flaky link if long-lived streams keep breaking
- `--publish`

  download volume data through the published (ingress) exporter endpoint instead of the in-cluster service; if unset, the in-cluster endpoint's reachability is auto-detected
- `--ttl`

  DataExport TTL (e.g. 2h, 30m)

- `--volume-compression`

  volume compression codec (none, zstd); block volumes: data.bin[.]; filesystem volumes: per-file compressed entries inside an uncompressed data.tar container
- `--volume-compression-level`

  compression level for the selected codec (0 = codec default; ignored when --volume-compression=none)
- `--workers`

  maximum number of nodes downloaded concurrently

d8 snapshot get

Aliases: `list`, `ls`

Usage:

```
d8 snapshot get [options]
```

List Snapshot resources in the cluster

Options

- `--all-namespaces`, `-A`

  list Snapshots across all namespaces
- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--namespace`, `-n`

  namespace to list Snapshots from (defaults to the kubeconfig context namespace)
- `--output`, `-o`

  Output format: table|json|yaml

d8 snapshot local

Usage:

```
d8 snapshot local [options]
```

Operate on a locally downloaded snapshot archive (no kubeconfig required)

d8 snapshot local describe

Usage:

```
d8 snapshot local describe <DIR> [options]
```

Show the structure of a locally downloaded snapshot archive as a tree

Show more...

Show the structure of a locally downloaded snapshot archive as a tree.

By default, every node's content checksum and authenticated snapshot.yaml metadata are verified before any output is rendered. Legacy archives without formatVersion and metadataChecksum are rejected because their identity and volume metadata are unauthenticated. Use --allow-unauthenticated-legacy only to inspect a trusted pre-version archive; the command emits a warning because compatibility mode cannot distinguish a genuine legacy archive from a downgraded and tampered current archive.

Options

- `--allow-unauthenticated-legacy`

  allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe compatibility mode)

d8 snapshot local get

Usage:

```
d8 snapshot local get <DIR> [options]
```

Print a one-line summary of a locally downloaded snapshot archive

Show more...

Print a one-line summary of a locally downloaded snapshot archive.

By default, every node's content checksum and authenticated snapshot.yaml metadata are verified before any output is rendered. Legacy archives without formatVersion and metadataChecksum are rejected because their identity and volume metadata are unauthenticated. Use --allow-unauthenticated-legacy only to inspect a trusted pre-version archive; the command emits a warning because compatibility mode cannot distinguish a genuine legacy archive from a downgraded and tampered current archive.

Options

- `--allow-unauthenticated-legacy`

  allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe compatibility mode)

d8 snapshot restore

Usage:

```
d8 snapshot restore [flags] <snapshot> [options]
```

Restore a snapshot in-place into its namespace

Show more...

Restore a snapshot tree in-namespace.

The snapshot is restored into the same namespace it lives in: -n/--namespace is both the source Snapshot namespace and the restore target. The required VolumeSnapshot / VirtualDiskSnapshot leaves must already be present in that namespace (they exist after the original snapshot, or after the snapshot is re-created in this namespace).

Cross-namespace restore is not a single command: it is a separate procedure that downloads the snapshot from the source namespace, re-creates the Snapshot and its volume-snapshot leaves in the target namespace, and then runs restore there.

The client walks the selected namespaced snapshot hierarchy and asks the server to compile one node at a time. Every node response is independently size-bounded, and the complete child-before-parent result is staged and validated before the first apply. PersistentVolumeClaims already carry spec.dataSourceRef pointing at the VolumeSnapshot (or VirtualDiskSnapshot for domain disks) present in the namespace, so CSI provisions the data.

--node restricts the restore to a single node subtree. The positional Snapshot always anchors the hierarchy and must exist; the selection may use either the generated snapshot-CR Kind/name or the original captured source Kind/name. A Ready child selected by its generated identity remains restorable when the root is DEGRADED because an unrelated sibling CR was deleted. Selecting that deleted generated child reports that it belongs to the tree but is missing. Original-source selection fails closed while any child ref is unresolved, because a missing child's original identity cannot be inspected; retry with the live match's generated Kind/name and --node-api-version shown in the error.

--node-api-version disambiguates nodes whose selected Kind/name is shared across API groups or versions. Use "v1" for the Kubernetes core group, or "<group>/<version>" for a named group. It requires --node and must match the generated or original identity used by that selection.

--scope narrows how much of the addressed node (the root, or --node's selection) the server compiles: "subtree" (default) compiles the node and its whole subtree, recursively; "node" compiles only the addressed node itself, with no descendants. --scope node with no --node selects the root Snapshot node alone.

--object restricts a --scope node restore to a single captured object within that node, by "<Kind>/<name>"; it requires --scope node and fails fast, before any network call, otherwise.

--dry-run sends every apply with DryRunAll so the API server validates and admits objects (schema validation, webhooks, immutable-field checks) without persisting them. Use it to preflight a restore before committing. The --wait loop is skipped in dry-run mode because no objects are actually created.

Without --edit or --dry-run, an interactive restore automatically opens one editor session when the pre-mutation DryRunAll pass returns Kubernetes Invalid (HTTP 422), including schema, webhook, and immutable-field Invalid responses. Both stdin and stdout must be terminals. Use --no-auto-edit to opt out. Conflict (HTTP 409), AlreadyExists, conflicting duplicate manifests, and an existing Bound PVC never trigger automatic editing. A generic Invalid response does not imply that the object already exists or that an immutable field differs.

After an automatic edit, restore restages and validates the complete manifest set and retries the full DryRunAll pass once; a repeated Invalid is returned without reopening the editor. Explicit --edit opens before preflight and suppresses later automatic editing. Explicit --dry-run never opens the automatic editor. In non-interactive use, Invalid is returned with the original API response preserved as its cause and without invoking an editor. Editor selection is $KUBE_EDITOR, then $EDITOR, falling back to vi; a blank selected command, failed editor, unchanged file, or empty file aborts instead of applying.

Before any dry-run or real apply, restore reads every target PersistentVolumeClaim. If a same-named claim is already Bound, restore refuses to reuse it because that status may describe stale data from an earlier operation. Restore never deletes or replaces the claim.

--wait only tracks PersistentVolumeClaims that appear in the restored manifest set, and it reports when this command's own wait is over, not when a workload can use the restored data. Disk-backed PVCs for domain objects are recreated asynchronously by the domain controller (not part of this output), so they are not awaited; the command may return before such volumes finish provisioning. A Pending PVC on a WaitForFirstConsumer StorageClass is not awaited while it has no selected node, live Pod consumer, active provisioning event, or terminal provisioning failure: every such claim is listed once, by namespace/name and StorageClass, as a claim this command does not wait to become Bound. Once selected, consumed, or provisioning, it is polled until Bound or --timeout; an observed provisioning failure is returned with its cause. An expired --timeout means only that the wait could not be confirmed within that period: the objects were already applied and are not rolled back. Success revalidates the post-apply PVC UID, restore source, and bound PersistentVolume identity; it does not promise that every restored PVC is Bound.

Options

- `--context`

  The name of the kubeconfig context to use
- `--dry-run`

  validate objects via DryRunAll without persisting; skips --wait (use to preflight a restore)
- `--edit`

  open resolved manifests in $KUBE_EDITOR/$EDITOR (falling back to vi) before applying; aborts on non-zero exit, unchanged, or empty content
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--namespace`, `-n`

  snapshot namespace; also the restore target namespace (required)
- `--no-auto-edit`

  disable the one automatic editor session for interactive Kubernetes Invalid (HTTP 422) preflight responses
- `--node`

  restrict restore to a single node subtree; format '/' (e.g. --node DemoVirtualDisk/bk-disk-a); the generated snapshot CR name form (e.g. DemoVirtualDiskSnapshot/nss-child-abc123) is still accepted
- `--node-api-version`

  disambiguate --node by exact API version: 'v1' for core resources or '/' for named groups (requires --node)
- `--object`

  restrict a --scope node restore to a single captured object; format '/' (requires --scope node)
- `--scope`

  restore scope: 'subtree' (default) compiles the addressed node and its whole subtree; 'node' compiles only the addressed node itself
- `--timeout`

  bounds the --wait Bound check only; an expired timeout does not roll back the applied objects
- `--wait`

  wait for applicable manifest PVCs to become Bound; a dormant WaitForFirstConsumer PVC with no selected node, live consumer, or provisioning signal is listed by name and is not awaited

d8 snapshot upload

Alias: `import`

Usage:

```
d8 snapshot upload [flags] [options]
```

Upload a downloaded snapshot archive into a namespace

Show more...

Upload a local snapshot archive (produced by 'd8 snapshot download') into a namespace.

The archive tree is reconstructed in stages: first every import-mode CR is created top-down (parents before children) so each child carries a child->parent ownerRef the state-snapshotter binders use to attach it to its parent; then every node's manifests plus direct child refs are uploaded to the state-snapshotter aggregated API; only after all manifests are uploaded are the data-leaf volume bytes uploaded via a DataImport. Manifests are uploaded before any volume data because a leaf's DataImport stays pending until the leaf VolumeSnapshot is bound, which requires its ancestors' manifests to already be present. After the whole tree is uploaded it waits for the root Snapshot and its bound SnapshotContent to become Ready, leaving the namespace ready for 'd8 snapshot restore'.

--node restricts the upload to a single node and its descendants. The selected node becomes the upload root; it must be a core Snapshot, a CSI VolumeSnapshot data leaf, or a domain data leaf (e.g. DemoVirtualDiskSnapshot). Domain aggregator nodes (a DemoVirtualMachineSnapshot that references child snapshots) and manifest-only domain nodes cannot be selected as the upload root (they have no parent SnapshotContent to attach to when uploaded standalone); they are uploaded only as part of a full-archive upload (omit --node) or by selecting an ancestor Snapshot.

Scope and limitations:
- Full Snapshot trees are uploaded as-is, including domain aggregator nodes (a DemoVirtualMachineSnapshot that aggregates child DemoVirtualDiskSnapshot nodes): the CLI creates each node's unified import marker and uploads its manifests + child refs, and the server-side genericbinder reconstructs the aggregator's content from its children's SnapshotContents. The aggregator is a non-root node in this case, so no DataImport is created for it (only its data-leaf descendants stream volume bytes).
- Manifest-only domain nodes (e.g. a disk-less DemoVirtualMachineSnapshot, which carries only manifests) are uploaded as part of their parent tree (the server materialises their content from the uploaded manifests alone, with no data leg).
- A domain aggregator can be reconstructed only within a tree, never as a standalone --node root. To upload an individual disk snapshot from such a tree on its own, use --node <DomainDataLeafKind>/<name> (e.g. --node DemoVirtualDiskSnapshot/dvd-1).
- Both block-volume and filesystem-volume data leaves are supported.
- Filesystem archives can upload regular files only. Symlink, hardlink, device, FIFO, and socket entries, plus empty directories other than well-known filesystem-reserved names (for example lost+found), cannot be uploaded: the protocol supports only regular-file PUTs, and creates directories only as a side effect of a file PUT inside them.
- --skip-unsupported-fs-entries uploads the supported regular files instead of failing, but causes data loss for each skipped path; review the bounded post-upload summary.
- Legacy archives without formatVersion and metadataChecksum are rejected by default because their snapshot.yaml identity and volume metadata are unauthenticated. Use --allow-unauthenticated-legacy only for deliberate migration or inspection of trusted pre-version archives; this mode cannot distinguish a genuine legacy archive from a downgraded and tampered current archive.
- Uploading requires RBAC to create DataImport (storage-volume-data-manager) and to call the manifests-and-children-refs-upload subresource (e.g. an admin kubeconfig); the read-only snapshot admin role is not sufficient.

--publish selects how each data leaf's bytes are streamed to its DataImport importer pod. With --publish=false (or when autodetection picks it), bytes go straight to the importer's in-cluster service, trusting only its internal CA (status.ca). With --publish=true, bytes go through the storage-foundation-published Ingress endpoint (status.publicURL) instead, so a kubeconfig without direct network access to the cluster's internal service network can still upload. If --publish is not given, the command probes whether the in-cluster importer endpoint is reachable and picks accordingly. IMPORTANT: the publish path works only with a kubeconfig authenticated by a bearer token. Ingress terminates TLS with its own certificate and does not forward the client's TLS certificate to the importer pod, so a certificate-based kubeconfig receives a 401 when --publish=true.

Options

- `--allow-existing`

  downgrade namespace preflight conflict check to a warning (import-mode markers from a prior run are never conflicts regardless of this flag)
- `--allow-unauthenticated-legacy`

  allow trusted pre-version archives whose snapshot.yaml metadata is unauthenticated (unsafe; explicit compatibility mode)
- `--context`

  The name of the kubeconfig context to use
- `--input`, `-i`

  root archive directory produced by 'd8 snapshot download' (required)
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

- `--namespace`, `-n`

  target namespace to import into (required)
- `--node`

  restrict import to a single node subtree; format '/' (e.g. --node VolumeSnapshot/pvc-1)
- `--publish`

  upload volume data through the published (ingress) importer endpoint instead of the in-cluster service; if unset, the in-cluster endpoint's reachability is auto-detected
- `--skip-unsupported-fs-entries`

  skip unsupported filesystem entries and report them after upload (causes data loss for skipped paths)
- `--timeout`

  timeout for per-node readiness/completion waits
- `--ttl`

  idle TTL for each data-leaf DataImport (e.g. 2h, 30m); must exceed the importer's provisioning and post-upload completion time
- `--workers`

  maximum number of data-leaf volume uploads to run in parallel (fixed cap via errgroup.SetLimit; default 5)

d8 status

Get cluster status information

Show more...

Get status information about Deckhouse Kubernetes Platform

© Flant JSC 2025

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 stronghold

Deckhouse Stronghold commands

d8 stronghold audit

Usage:

```
d8 stronghold [command] [args...] audit [options]
```

Interact with audit devices

d8 stronghold audit disable

Usage:

```
d8 stronghold [command] [args...] audit disable [options]
```

Disables an audit device

d8 stronghold audit enable

Usage:

```
d8 stronghold [command] [args...] audit enable [options]
```

Enables an audit device

d8 stronghold audit list

Usage:

```
d8 stronghold [command] [args...] audit list [options]
```

Lists enabled audit devices

d8 stronghold auth

Usage:

```
d8 stronghold [command] [args...] auth [options]
```

Interact with auth methods

d8 stronghold auth disable

Usage:

```
d8 stronghold [command] [args...] auth disable [options]
```

Disables an auth method

d8 stronghold auth enable

Usage:

```
d8 stronghold [command] [args...] auth enable [options]
```

Enables a new auth method

d8 stronghold auth help

Usage:

```
d8 stronghold [command] [args...] auth help [options]
```

Prints usage for an auth method

d8 stronghold auth list

Usage:

```
d8 stronghold [command] [args...] auth list [options]
```

Lists enabled auth methods

d8 stronghold auth move

Usage:

```
d8 stronghold [command] [args...] auth move [options]
```

Move an auth method to a new path

d8 stronghold auth tune

Usage:

```
d8 stronghold [command] [args...] auth tune [options]
```

Tunes an auth method configuration

d8 stronghold debug

Usage:

```
d8 stronghold [command] [args...] debug [options]
```

Runs the debug command

d8 stronghold delete

Usage:

```
d8 stronghold [command] [args...] delete [options]
```

Delete secrets and configuration

d8 stronghold events

Usage:

```
d8 stronghold [command] [args...] events [options]
```

d8 stronghold events subscribe

Usage:

```
d8 stronghold [command] [args...] events subscribe [options]
```

Subscribe to events

d8 stronghold kv

Usage:

```
d8 stronghold [command] [args...] kv [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv delete

Usage:

```
d8 stronghold [command] [args...] kv delete [options]
```

Deletes versions in the KV store

d8 stronghold kv destroy

Usage:

```
d8 stronghold [command] [args...] kv destroy [options]
```

Permanently removes one or more versions in the KV store

d8 stronghold kv enable-versioning

Usage:

```
d8 stronghold [command] [args...] kv enable-versioning [options]
```

Turns on versioning for a KV store

d8 stronghold kv get

Usage:

```
d8 stronghold [command] [args...] kv get [options]
```

Retrieves data from the KV store

d8 stronghold kv list

Usage:

```
d8 stronghold [command] [args...] kv list [options]
```

List data or secrets

d8 stronghold kv metadata

Usage:

```
d8 stronghold [command] [args...] kv metadata [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv metadata delete

Usage:

```
d8 stronghold [command] [args...] kv metadata delete [options]
```

Deletes all versions and metadata for a key in the KV store

d8 stronghold kv metadata get

Usage:

```
d8 stronghold [command] [args...] kv metadata get [options]
```

Retrieves key metadata from the KV store

d8 stronghold kv metadata patch

Usage:

```
d8 stronghold [command] [args...] kv metadata patch [options]
```

Patches key settings in the KV store

d8 stronghold kv metadata put

Usage:

```
d8 stronghold [command] [args...] kv metadata put [options]
```

Sets or updates key settings in the KV store

d8 stronghold kv patch

Usage:

```
d8 stronghold [command] [args...] kv patch [options]
```

Sets or updates data in the KV store without overwriting

d8 stronghold kv put

Usage:

```
d8 stronghold [command] [args...] kv put [options]
```

Sets or updates data in the KV store

d8 stronghold kv rollback

Usage:

```
d8 stronghold [command] [args...] kv rollback [options]
```

Rolls back to a previous version of data

d8 stronghold kv undelete

Usage:

```
d8 stronghold [command] [args...] kv undelete [options]
```

Undeletes versions in the KV store

d8 stronghold lease

Usage:

```
d8 stronghold [command] [args...] lease [options]
```

Interact with leases

d8 stronghold lease lookup

Usage:

```
d8 stronghold [command] [args...] lease lookup [options]
```

Lookup the lease of a secret

d8 stronghold lease renew

Usage:

```
d8 stronghold [command] [args...] lease renew [options]
```

Renews the lease of a secret

d8 stronghold lease revoke

Usage:

```
d8 stronghold [command] [args...] lease revoke [options]
```

Revokes leases and secrets

d8 stronghold list

Usage:

```
d8 stronghold [command] [args...] list [options]
```

List data or secrets

d8 stronghold login

Usage:

```
d8 stronghold [command] [args...] login [options]
```

Authenticate locally

d8 stronghold monitor

Usage:

```
d8 stronghold [command] [args...] monitor [options]
```

Stream log messages from a Stronghold server

d8 stronghold namespace

Usage:

```
d8 stronghold [command] [args...] namespace [options]
```

Interact with namespaces

d8 stronghold namespace create

Usage:

```
d8 stronghold [command] [args...] namespace create [options]
```

Create a new namespace

d8 stronghold namespace delete

Usage:

```
d8 stronghold [command] [args...] namespace delete [options]
```

Delete an existing namespace

d8 stronghold namespace list

Usage:

```
d8 stronghold [command] [args...] namespace list [options]
```

List child namespaces

d8 stronghold namespace lock

Usage:

```
d8 stronghold [command] [args...] namespace lock [options]
```

Lock the API for particular namespaces

d8 stronghold namespace lookup

Usage:

```
d8 stronghold [command] [args...] namespace lookup [options]
```

Look up an existing namespace

d8 stronghold namespace patch

Usage:

```
d8 stronghold [command] [args...] namespace patch [options]
```

Patch an existing namespace

d8 stronghold namespace unlock

Usage:

```
d8 stronghold [command] [args...] namespace unlock [options]
```

Unlock the API for particular namespaces

d8 stronghold operator

Usage:

```
d8 stronghold [command] [args...] operator [options]
```

Perform operator-specific tasks

d8 stronghold operator generate-root

Usage:

```
d8 stronghold [command] [args...] operator generate-root [options]
```

Generates a new root, DR operation, or recovery token

d8 stronghold operator init

Usage:

```
d8 stronghold [command] [args...] operator init [options]
```

Initializes a server

d8 stronghold operator key-status

Usage:

```
d8 stronghold [command] [args...] operator key-status [options]
```

Provides information about the active encryption key

d8 stronghold operator members

Usage:

```
d8 stronghold [command] [args...] operator members [options]
```

Returns the list of nodes in the cluster

d8 stronghold operator raft

Usage:

```
d8 stronghold [command] [args...] operator raft [options]
```

Interact with Vault's raft storage backend

d8 stronghold operator raft autopilot

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot [options]
```

d8 stronghold operator raft autopilot get-config

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot get-config [options]
```

Returns the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot set-config

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot set-config [options]
```

Modify the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot state

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot state [options]
```

Displays the state of the raft cluster under integrated storage as seen by autopilot

d8 stronghold operator raft demote

Usage:

```
d8 stronghold [command] [args...] operator raft demote [options]
```

Demotes a voter to a permanent non-voter

d8 stronghold operator raft join

Usage:

```
d8 stronghold [command] [args...] operator raft join [options]
```

Joins a node to the Raft cluster

d8 stronghold operator raft list-peers

Usage:

```
d8 stronghold [command] [args...] operator raft list-peers [options]
```

Returns the Raft peer set

d8 stronghold operator raft promote

Usage:

```
d8 stronghold [command] [args...] operator raft promote [options]
```

Promotes a permanent non-voter to a voter

d8 stronghold operator raft remove-peer

Usage:

```
d8 stronghold [command] [args...] operator raft remove-peer [options]
```

Removes a node from the Raft cluster

d8 stronghold operator raft snapshot

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot [options]
```

Restores, saves and inspects snapshots from the Raft cluster

d8 stronghold operator raft snapshot inspect

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot inspect [options]
```

Inspects a Raft snapshot

d8 stronghold operator raft snapshot restore

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot restore [options]
```

Installs the provided snapshot, returning the cluster to the state defined in it

d8 stronghold operator raft snapshot save

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot save [options]
```

Saves a snapshot of the current state of the Raft cluster into a file

d8 stronghold operator rekey

Usage:

```
d8 stronghold [command] [args...] operator rekey [options]
```

Generates new unseal keys

d8 stronghold operator rotate

Usage:

```
d8 stronghold [command] [args...] operator rotate [options]
```

Rotates the underlying encryption key

d8 stronghold operator seal

Usage:

```
d8 stronghold [command] [args...] operator seal [options]
```

Seals the Stronghold server

d8 stronghold operator step-down

Usage:

```
d8 stronghold [command] [args...] operator step-down [options]
```

Forces Stronghold to resign active duty

d8 stronghold operator unseal

Usage:

```
d8 stronghold [command] [args...] operator unseal [options]
```

Unseals the Stronghold server

d8 stronghold operator usage

Usage:

```
d8 stronghold [command] [args...] operator usage [options]
```

Lists historical client counts

d8 stronghold patch

Usage:

```
d8 stronghold [command] [args...] patch [options]
```

Patch data, configuration, and secrets

d8 stronghold path-help

Usage:

```
d8 stronghold [command] [args...] path-help [options]
```

Retrieve API help for paths

d8 stronghold pki

Usage:

```
d8 stronghold [command] [args...] pki [options]
```

Interact with Stronghold's PKI Secrets Engine

d8 stronghold pki health-check

Usage:

```
d8 stronghold [command] [args...] pki health-check [options]
```

Check a PKI Secrets Engine mount's health and operational status

d8 stronghold pki issue

Usage:

```
d8 stronghold [command] [args...] pki issue [options]
```

Given a parent certificate, and a list of generation parameters, creates an issuer on a specified mount

d8 stronghold pki list-intermediates

Usage:

```
d8 stronghold [command] [args...] pki list-intermediates [options]
```

Determine which of a list of certificates, were issued by a given parent certificate

d8 stronghold pki reissue

Usage:

```
d8 stronghold [command] [args...] pki reissue [options]
```

Uses a parent certificate and a template certificate to create a new issuer on a child mount

d8 stronghold pki verify-sign

Usage:

```
d8 stronghold [command] [args...] pki verify-sign [options]
```

Check whether one certificate validates another specified certificate

d8 stronghold plugin

Usage:

```
d8 stronghold [command] [args...] plugin [options]
```

Interact with Stronghold plugins and catalog

d8 stronghold plugin deregister

Usage:

```
d8 stronghold [command] [args...] plugin deregister [options]
```

Deregister an existing plugin in the catalog

d8 stronghold plugin info

Usage:

```
d8 stronghold [command] [args...] plugin info [options]
```

Read information about a plugin in the catalog

d8 stronghold plugin list

Usage:

```
d8 stronghold [command] [args...] plugin list [options]
```

Lists available plugins

d8 stronghold plugin register

Usage:

```
d8 stronghold [command] [args...] plugin register [options]
```

Registers a new plugin in the catalog

d8 stronghold plugin reload

Usage:

```
d8 stronghold [command] [args...] plugin reload [options]
```

Reload mounted plugin backend

d8 stronghold plugin reload-status

Usage:

```
d8 stronghold [command] [args...] plugin reload-status [options]
```

Get the status of an active or recently completed global plugin reload

d8 stronghold policy

Usage:

```
d8 stronghold [command] [args...] policy [options]
```

Interact with policies

d8 stronghold policy delete

Usage:

```
d8 stronghold [command] [args...] policy delete [options]
```

Deletes a policy by name

d8 stronghold policy list

Usage:

```
d8 stronghold [command] [args...] policy list [options]
```

Lists the installed policies

d8 stronghold policy read

Usage:

```
d8 stronghold [command] [args...] policy read [options]
```

Prints the contents of a policy

d8 stronghold policy write

Usage:

```
d8 stronghold [command] [args...] policy write [options]
```

Uploads a named policy from a file

d8 stronghold print

Usage:

```
d8 stronghold [command] [args...] print [options]
```

Prints runtime configurations

d8 stronghold print token

Usage:

```
d8 stronghold [command] [args...] print token [options]
```

Prints token currently in use

d8 stronghold read

Usage:

```
d8 stronghold [command] [args...] read [options]
```

Read data and retrieves secrets

d8 stronghold secrets

Usage:

```
d8 stronghold [command] [args...] secrets [options]
```

Interact with secrets engines

d8 stronghold secrets disable

Usage:

```
d8 stronghold [command] [args...] secrets disable [options]
```

Disable a secret engine

d8 stronghold secrets enable

Usage:

```
d8 stronghold [command] [args...] secrets enable [options]
```

Enable a secrets engine

d8 stronghold secrets list

Usage:

```
d8 stronghold [command] [args...] secrets list [options]
```

List enabled secrets engines

d8 stronghold secrets move

Usage:

```
d8 stronghold [command] [args...] secrets move [options]
```

Move a secrets engine to a new path

d8 stronghold secrets tune

Usage:

```
d8 stronghold [command] [args...] secrets tune [options]
```

Tune a secrets engine configuration

d8 stronghold ssh

Usage:

```
d8 stronghold [command] [args...] ssh [options]
```

Initiate an SSH session

d8 stronghold status

Usage:

```
d8 stronghold [command] [args...] status [options]
```

Print seal and HA status

d8 stronghold token

Usage:

```
d8 stronghold [command] [args...] token [options]
```

Interact with tokens

d8 stronghold token capabilities

Usage:

```
d8 stronghold [command] [args...] token capabilities [options]
```

Print capabilities of a token on a path

d8 stronghold token create

Usage:

```
d8 stronghold [command] [args...] token create [options]
```

Create a new token

d8 stronghold token lookup

Usage:

```
d8 stronghold [command] [args...] token lookup [options]
```

Display information about a token

d8 stronghold token renew

Usage:

```
d8 stronghold [command] [args...] token renew [options]
```

Renew a token lease

d8 stronghold token revoke

Usage:

```
d8 stronghold [command] [args...] token revoke [options]
```

Revoke a token and its children

d8 stronghold transit

Usage:

```
d8 stronghold [command] [args...] transit [options]
```

Interact with Stronghold's Transit Secrets Engine

d8 stronghold transit import

Usage:

```
d8 stronghold [command] [args...] transit import [options]
```

Import a key into the Transit secrets engines.

d8 stronghold transit import-version

Usage:

```
d8 stronghold [command] [args...] transit import-version [options]
```

Import key material into a new key version in the Transit secrets engines.

d8 stronghold unwrap

Usage:

```
d8 stronghold [command] [args...] unwrap [options]
```

Unwrap a wrapped secret

d8 stronghold version

Usage:

```
d8 stronghold [command] [args...] version [options]
```

Prints the Stronghold CLI version

d8 stronghold version-history

Usage:

```
d8 stronghold [command] [args...] version-history [options]
```

Prints the version history of the target Stronghold server

d8 stronghold write

Usage:

```
d8 stronghold [command] [args...] write [options]
```

Write data, configuration, and secrets

d8 system

Aliases: `s`, `p`, `platform`

Operate system options. Aliases: s, p, platform

Show more...

Operate system options in DKP.

© Flant JSC 2025

Options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system collect-debug-info

Usage:

```
d8 system collect-debug-info [flags] > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz [options]
```

Collect debug info.

Show more...

Collect debug info from Deckhouse Kubernetes Platform.

© Flant JSC 2025

Options

- `--command-timeout`
  Timeout for each individual debug command execution
- `--exclude`
  Exclude specific files from the debug archive. Use comma-separated names as printed by --list-exclude; the file extension is optional. An unknown name is an error
- `--list-exclude`, `-l`
  List all files that can be excluded from the debug archive, then exit
- `--request-interval`
  Minimum interval between debug command executions to avoid overloading the cluster (e.g. 200ms, 500ms, 1s). Zero disables rate limiting (default 0s)

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system collect-debug-info virtualization

Usage:

```
d8 system collect-debug-info [flags] > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz virtualization [flags] >
deckhouse-debug-virtualization-$(date +"%Y_%m_%d").tar.gz [options]
```

Collect a separate virtualization debug archive.

Show more...

Collect a separate debug archive with detailed data from the d8-virtualization namespace.

Options

- `--command-timeout`

  Timeout for each individual debug command execution
- `--request-interval`

  Minimum interval between debug command executions to avoid overloading the cluster (e.g. 200ms, 500ms, 1s). Zero disables rate limiting (default 0s)
- `--skip-ds-logs`

  Skip collecting logs from pods managed by a DaemonSet (virt-handler, virtualization-dra, vm-route-forge, ...) to reduce archive size on clusters with many nodes

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system edit

Usage:

```
d8 system edit [options]
```

Edit configuration files

Show more...

Change configuration files in Kubernetes cluster conveniently and safely.

© Flant JSC 2025

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system edit cluster-configuration

Usage:

```
d8 system edit cluster-configuration [options]
```

Edit cluster-configuration.

Show more...

Edit cluster-configuration in Kubernetes cluster.

© Flant JSC 2025

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system edit provider-cluster-configuration

Usage:

```
d8 system edit provider-cluster-configuration [options]
```

Edit provider-cluster-configuration.

Show more...

Edit provider-cluster-configuration in Kubernetes cluster.

© Flant JSC 2025

Options

- `--editor`, `-e`
  Your favourite editor.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system edit static-cluster-configuration

Usage:

```
d8 system edit static-cluster-configuration [options]
```

Edit static-cluster-configuration.

Show more...

Edit static-cluster-configuration in Kubernetes cluster.

© Flant JSC 2025

Options

- `--editor`, `-e`
  Your favourite editor.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system get

Usage:

```
d8 system get [options]
```

Get configuration files

Show more...

Get configuration files from Kubernetes cluster.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for

the current OS user)

d8 system get cluster-configuration

Usage:

```
d8 system get cluster-configuration [options]
```

Get cluster-configuration.

Show more...

Get cluster-configuration from Kubernetes cluster.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system get provider-cluster-configuration

Usage:

```
d8 system get provider-cluster-configuration [options]
```

Get provider-cluster-configuration.

Show more...

Get provider-cluster-configuration from Kubernetes cluster.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system get static-cluster-configuration

Usage:

```
d8 system get static-cluster-configuration [options]
```

Get static-cluster-configuration.

Show more...

Get static-cluster-configuration from Kubernetes cluster.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system logs

Usage:

```
d8 system logs [options]
```

Inspect deckhouse-controller logs.

Show more...

Inspect deckhouse-controller logs from Deckhouse Kubernetes Platform.

© Flant JSC 2025

Options

- `--follow`, `-f`
  Specify if the logs should be streamed.
- `--since`
  Show logs newer than a relative duration like 5s, 2m, or 1h. Only one of since-time / since may be used.
- `--since-time`
  Show logs after a specific timestamp, e.g. --since-time='2025-05-19 12:00:00'. Only one of since-time / since may be used.
- `--tail`
  Limit the number of lines in the output to the last N lines. Defaults to -1 with no limit.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module

Usage:

```
d8 system module [options]
```

Operate the Deckhouse Kubernetes Platform modules

Show more...

Operate the Deckhouse Kubernetes Platform modules.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module apply-now

Usage:

```
d8 system module apply-now <module-name> <version> [options]
```

Apply a module release immediately, bypassing update windows.

Show more...

Apply a module release immediately, bypassing update windows.

This command adds the annotation 'modules.deckhouse.io/apply-now="true"' to the specified ModuleRelease resource, forcing immediate deployment regardless of configured update windows or applyAfter schedules.

This command is used for modules with Auto update policy that have update windows configured or releases scheduled for future deployment.

© Flant JSC 2025

Global options

- `--context`

The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module approve

Usage:

```
d8 system module approve <module-name> <version> [options]
```

Approve a pending module release for Manual update policy.

Show more...

Approve a pending module release for installation.

This command adds the annotation 'modules.deckhouse.io/approved="true"' to the specified ModuleRelease resource, allowing it to be deployed according to the update policy.

This command is used for modules with Manual update policy that require explicit approval before deployment.

© Flant JSC 2025

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module disable

Usage:

```
d8 system module disable [options]
```

Disable module.

Show more...

Disable module using the ModuleConfig resource.

© Flant JSC 2025

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module enable

Usage:

```
d8 system module enable [options]
```

Enable module.

Show more...

Enable module using the ModuleConfig resource.

© Flant JSC 2025

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module list

Usage:

```
d8 system module list [options]
```

List enabled modules.

Show more...

List enabled Deckhouse Kubernetes Platform modules.

© Flant JSC 2025

Options

- `--output`, `-o`
  Output format: yaml|json

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module maintenance

Usage:

```
d8 system module maintenance [options]
```

Enable or disable module maintenance mode.

Show more...

Enable or disable maintenance mode for a module via its ModuleConfig resource.

While maintenance mode is enabled, Deckhouse stops reconciling the module's resources (it sets 'spec.maintenance="NoResourceReconciliation"' on the ModuleConfig). This lets you develop or tweak the module's resources.

Disabling maintenance mode removes the 'spec.maintenance' field and lets Deckhouse resume normal reconciliation.

The module's ModuleConfig must already exist. If it does not enable it first.

© Flant JSC 2026

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module maintenance disable

Usage:

```
d8 system module maintenance disable <module-name> [options]
```

Disable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module maintenance enable

Usage:

```
d8 system module maintenance enable <module-name> [options]
```

Enable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module snapshots

Usage:

```
d8 system module snapshots [options]
```

Dump shapshots.

Show more...

Dump module hooks snapshots.

© Flant JSC 2025

Options

- `--output`, `-o`
  Output format: yaml|json

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system module values

Usage:

```
d8 system module values [options]
```

Dump values.

Show more...

Dump module hooks values.

© Flant JSC 2025

Options

- `--output`, `-o`
  Output format: yaml|json

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system package

Usage:

```
d8 system package [options]
```

Operate the Deckhouse Kubernetes Platform packages

Show more...

Operate the Deckhouse Kubernetes Platform packages.

© Flant JSC 2026

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system package scan

Usage:

```
d8 system package scan <repository-name> [options]
```

Create a scan task for a PackageRepository

Show more...

Create a scan task for a PackageRepository.

This command creates a PackageRepositoryOperation resource that triggers a full scan of the specified package repository.

© Flant JSC 2026

Options

- `--dry-run`
  Preview the operation without creating it
- `--name`
  Custom name for the PackageRepositoryOperation (auto-generated if not specified)
- `--timeout`
  Timeout for the scan operation

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system queue

Usage:

```
d8 system queue [options]
```

Dump queues.

Show more...

Dump queues from Deckhouse Kubernetes Platform.

© Flant JSC 2025

Global options

- `--context`
  The name of the kubeconfig context to use

- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system queue list

Usage:

```
d8 system queue list [options]
```

Dump all queues.

Show more...

Dump all Deckhouse Kubernetes Platform queues.

© Flant JSC 2025

Options

- `--http`

  Query the Deckhouse debug server through a Kubernetes API port-forward instead of exec into the pod. Needs pods/portforward, not pods/exec.
- `--output`, `-o`

  Output format: json|yaml|text.
- `--show-empty`, `-e`

  Show empty queues.
- `--watch`, `-w`

  Watch for changes to the queue list.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 system queue main

Usage:

```
d8 system queue main [options]
```

Dump main queue.

Show more...

Dump main Deckhouse Kubernetes Platform queues.

© Flant JSC 2025

Options

- `--http`

  Query the Deckhouse debug server through a Kubernetes API port-forward instead of exec into the pod. Needs pods/portforward, not pods/exec.
- `--output`, `-o`

  Output format: json|yaml|text.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  Path to kubeconfig file. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)

d8 tools

Alias: `t`

Various useful tools for operating in The Deckhouse Ecosystem.

Show more...

Various useful tools for operating in The Deckhouse Ecosystem.

© Flant JSC 2025

d8 tools cosign

Usage:

```
d8 tools cosign [options]
```

Sign and verify container images with cosign v2

Show more...

Built-in cosign v2 for signing and verifying container images.

Deckhouse image signature verification (admission-policy-engine) supports cosign v2 signatures only, cosign v3 is not supported.

Run "d8 tools cosign --help" for the full list of cosign commands.

© Flant JSC 2026

d8 tools far-convert

Usage:

```
d8 tools far-convert [options]
```

Converts files with Falco rules to FalcoAuditRules CRD format

Show more...

Converts files with Falco rules to FalcoAuditRules CRD format.

© Flant JSC 2025

d8 tools gostsum

Usage:

```
d8 tools gostsum [options]
```

Calculate Streebog checksum of the data according to the RFC 6986 (GOST R 34.11-2012) specifications

Show more...

Calculate Streebog checksum of the data according to the RFC 6986 (GOST R 34.11-2012) specifications.

Data for checksumming is provided as a list of filepaths in the command-line arguments. If no filepaths are provided, data is assumed to be available from stdin.

© Flant JSC 2025

Options

- `--hash-length`, `-l`

  Bit-length variant of the algorithm to use. Only 256 (default) and 512 bit lengths are supported.

d8 tools htpasswd

Usage:

```
d8 tools htpasswd [flags] [passwordfile] [username] [password] [options]
```

Manage password files and hash passwords (Apache htpasswd analog)

Show more...

Manage password files and hash passwords, a self-contained analog of Apache htpasswd. No external htpasswd binary is required.

Usage mirrors htpasswd:

```
d8 tools htpasswd [-cbdps... -C cost -r rounds] passwordfile username
d8 tools htpasswd -b [...] passwordfile username password
d8 tools htpasswd -n [-bmBdps...] [username]
d8 tools htpasswd -D passwordfile username
d8 tools htpasswd -v passwordfile username
```

Algorithms (choose one; default bcrypt):

```
-B  bcrypt        secure, the default
-m  apr1 MD5      Apache MD5, legacy htpasswd default
-2  SHA-256 crypt  secure
-5  SHA-512 crypt  secure
-d  CRYPT (DES)    INSECURE, only first 8 chars used
-s  SHA-1          INSECURE, unsalted
-p  plaintext      INSECURE, no hashing
```

Unlike Apache htpasswd (apr1 at cost 5 by default), d8 defaults to bcrypt at cost 10, so 'd8 tools htpasswd -n <username>' produces a strong hash ready for 'd8 iam user create/reset-password --password-hash'. With -n and no username the bare hash is printed, which is exactly what --password-hash expects. The -2 and -5 SHA-crypt algorithms and the -r rounds flag are d8 extensions; Apache htpasswd has no -2, -5, or -r flag.

© Flant JSC 2026

Options

- `--batch`, `-b`

  Batch mode: take the password from the command line.
- `--bcrypt`, `-B`

  Use bcrypt (secure; the default).
- `--cost`, `-C`

  bcrypt cost/work factor (4-31); only with -B.
- `--create`, `-c`

  Create a new password file, overwriting any existing one.
- `--delete`, `-D`

  Delete the given user from the password file.
- `--des`, `-d`

  Use CRYPT (DES; INSECURE, 8-char limit).
- `--md5`, `-m`

  Use Apache MD5 (apr1).
- `--plaintext`, `-p`

  Store the password in plaintext (INSECURE).
- `--rounds`, `-r`

  SHA-256/512 rounds (1000-999999999); only with -2/-5.
- `--sha1`, `-s`

  Use SHA-1 (INSECURE, unsalted).
- `--sha256`, `-2`

  Use SHA-256 crypt (secure).
- `--sha512`, `-5`

  Use SHA-512 crypt (secure).
- `--stdin`, `-i`

  Read the password from stdin without verification.
- `--stdout`, `-n`

  Do not update a file; print the result to stdout.
- `--verify`, `-v`

  Verify the given password for the user.

d8 tools imagedigest

Usage:

```
d8 tools imagedigest [options]
```

Options

- `--debug`, `-d`

Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest add

Usage:

```
d8 tools imagedigest add <image> [options]
```

Calculating and adding the image digest to the image metadata according to the GOST standard Streebog (GOST R 34.11-2012)

Show more...

Calculating and adding the image digest to the image metadata according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest calculate

Usage:

```
d8 tools imagedigest calculate <image> [options]
```

Calculating the image digest according to the GOST standard Streebog (GOST R 34.11-2012)

Show more...

Calculating the image digest according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest calculate-from-file

Usage:

```
d8 tools imagedigest calculate-from-file <file> [options]
```

Calculating the file digest according to the GOST standard Streebog (GOST R 34.11-2012). For stdin use '-'

Show more...

Calculating the file digest according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest validate

Usage:

```
d8 tools imagedigest validate <image> [options]
```

Validating the image digest in the image metadata calculated according to the GOST standard Streebog (GOST R 34.11-2012)

Show more...

Validating the image digest in the image metadata calculated according to the GOST standard Streebog (GOST R 34.11-2012)

Options

- `--fix`

  Fix GOST Image Digest if it is incorrect

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools pki

Usage:

```
d8 tools pki [options]
```

Tools for working with control-plane PKI

Show more...

Tools for working with control-plane PKI.

© Flant JSC 2026

d8 tools pki certs

Usage:

```
d8 tools pki certs [options]
```

Manage and inspect control-plane TLS certificates

Show more...

Manage and inspect control-plane TLS certificates.

© Flant JSC 2026

d8 tools pki certs check

Usage:

```
d8 tools pki certs check [PATH] [options]
```

Check expiration of control-plane certificates

Show more...

Check expiration of control-plane certificates and kubeconfig client certificates.

Without arguments, all known control-plane certificates under --path and the kubeconfig files under --kubeconfig-dir are checked. Output is split into two sections – leaf certificates and certificate authorities – similar to "kubeadm certs check-expiration".

Missing known files are skipped in full-scan mode, so on worker or arbiter nodes the command prints the artifacts that are actually present locally.

--kubeconfig-dir defaults to the parent directory of --path, which matches the standard Kubernetes layout

(/etc/kubernetes/pki → /etc/kubernetes). Set it explicitly when using a non-default directory layout.

With a single PATH argument, only that file is inspected. The command auto-detects whether the file is a kubeconfig or a PEM certificate.

© Flant JSC 2026

Options

- `--kubeconfig-dir`

  Directory containing kubeconfig files (full-scan mode only; defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates (used in full-scan mode only)

d8 tools pki certs renew

Usage:

```
d8 tools pki certs renew (all | PATH) [options]
```

Renew control-plane certificates and kubeconfig client certificates

Show more...

Renew control-plane certificates and kubeconfig client certificates.

Renewal is unconditional — certificates are re-signed regardless of their current expiration date.

All Subject/SAN/Usage fields are read from the existing certificate on disk. No cluster configuration is required — this command is designed for emergency recovery when the Kubernetes API is unavailable.

The signing CA must be present on disk and must not be expired. If the CA private key is absent (external CA), or the CA certificate has expired, the certificate is skipped; renewing leaf certificates against an expired CA is pointless because chain validation will fail. The command exits non-zero if any certificate was skipped.

After renewal, restart kube-apiserver, kube-controller-manager, kube-scheduler and etcd so that the new certificates take effect or reboot the node/kubelet.

© Flant JSC 2026

Options

- `--dry-run`

  Run all without writing any files
- `--kubeconfig-dir`

  Directory containing kubeconfig files (defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates and CA files

d8 tools pki certs renew all

Usage:

```
d8 tools pki certs renew (all | PATH) all [options]
```

Renew all control-plane certificates and kubeconfig client certificates

Options

- `--dry-run`

  Run all without writing any files
- `--kubeconfig-dir`

  Directory containing kubeconfig files (defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates and CA files
- `--san`

  New IP SAN to add to serving certificates (e.g. the new master IP)

d8 tools sig-migrate

Usage:

```
d8 tools sig-migrate [options]
```

Migrate Kubernetes resources by adding and removing migration annotations

Show more...

Migrate Kubernetes resources by adding and removing migration annotations.

This command collects all Kubernetes resources (both namespaced and cluster-wide), adds a migration annotation (d8-migration=<timestamp>), and then removes the d8-migration- annotation prefix. This is useful for triggering reconciliation in Deckhouse controllers.

The command supports: - Retrying failed annotations from previous runs - Using different service accounts via impersonation - Configurable log levels (INFO, DEBUG, TRACE)

© Flant JSC 2025

Options

- `--as`

  Specify a Kubernetes service account for the kubectl operations (impersonation).
- `--context`

  The name of the kubeconfig context to use.
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests. (default is $KUBECONFIG when it is set, otherwise the default kubeconfig path for the current OS user)
- `--log-level`

  Set the log level (INFO, DEBUG, TRACE). Defaults to DEBUG.
- `--measure-stages`

  Print execution time for major migration stages.
- `--object`

  Process objects by identifier in format //. Use 'clusterwide' namespace for cluster-scoped resources. Resource name must match kubectl api-resources output.
- `--retry`

  Retry annotation for previously failed objects.
- `--threads`

  Number of worker threads for resource discovery and migration. Values <=0 use default.

d8 v

Alias: `virtualization`

Commands to work with Deckhouse Virtualization Platform.

Options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections

insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ansible-inventory

Usage:

```
d8 v [command] [args...] ansible-inventory [options]
```

Generate ansible inventory from virtual machines (ansible inventory script compatible).

Show more...

Generate ansible inventory from virtual machines.

When called without arguments, returns all hosts (same as --list).

Only virtual machines with assigned IP addresses are included in the inventory.

Arguments:
  --list   Return all hosts (default behavior if no arguments)
  --host <hostname>   Show variables for a particular host; output format matches inventory entries
  --output, -o <format> Output format: json, ini, or yaml (default: json)
  --namespace, -n <ns>   Namespace to list virtual machines from
                        (overrides kubeconfig context namespace)

Host names format: <vmname>.<namespace> (e.g., myvm.default)

VM annotations:
  - Annotations with prefix 'vars.ansible.deckhouse.io/' are included
    as host variables (prefix is stripped from variable name)
  - Use 'ansible.deckhouse.io/groups' annotation to add VMs to groups
    (comma-separated list of group names)

Network access:
  - For VM access, the Default network interface is used
  - The 'ansible_ssh_common_args' variable is automatically set for port-forwarding
    through kubectl using 'd8 v port-forward' command; the --context, --server and
    --kubeconfig flags of this command are carried over into it
  - Additional network interfaces are not currently supported

Options

- `--host`

  Return variables for specific host
- `--list`

  Return all hosts (default behavior if no arguments)
- `--namespace`, `-n`

  Namespace to list virtual machines from (overrides kubeconfig context namespace)
- `--output`, `-o`

  Output format: json, ini, or yaml (default: json)

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log

format)

d8 v collect-debug-info

Usage:

```
d8 v [command] [args...] collect-debug-info (VirtualMachine) [options]
```

Collect debug information for VM: configuration, events, and logs. Output is written as compressed archive to stdout.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v console

Usage:

```
d8 v [command] [args...] console (VirtualMachine) [options]
```

Connect to a console of a virtual machine.

Options

- `--force`
  Connect without asking, even when somebody else is using the console.
- `--timeout`
  Duration to wait until console is successfully connected (e.g., 1s, 5m, 300s).

Global options

- `--as`
  Username to impersonate for the operation
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--proxy-url`
  If provided, this URL will be used to connect via proxy
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
  Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v evict

Usage:

```
d8 v [command] [args...] evict (VirtualMachine) [options]
```

Evict a virtual machine.

Options

- `--all`

  Set this flag to apply the action to all VMs.
- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--label-selector`, `-l`

  Set this flag to apply the action to VMs with the specified labels.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.
- `--yes`, `-y`

  Set this flag to confirm the action without prompting for confirmation.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v migrate

Usage:

```
d8 v [command] [args...] migrate (VirtualMachine) [options]
```

Migrate a virtual machine.

Options

- `--all`

  Set this flag to apply the action to all VMs.
- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--label-selector`, `-l`

  Set this flag to apply the action to VMs with the specified labels.
- `--target-node-name`

  Set the target node name for virtual machine migration.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.
- `--yes`, `-y`

  Set this flag to confirm the action without prompting for confirmation.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v port-forward

Usage:

```
d8 v [command] [args...] port-forward name[.namespace] [protocol/]localPort[:targetPort]... [options]
```

Forward local ports to a virtual machine

Show more...

Forward local ports to a virtualmachine.
The port argument supports the syntax protocol/localPort:targetPort with protocol/ and :targetPort as optional fields.
Protocol supports TCP (default) and UDP.

Portforwards get established over the Kubernetes control-plane using websocket streams.
Usage can be restricted by the cluster administrator through the /portforward subresource.

Options

- `--address`

--address=: Set this to the address the local ports should be opened on

- `--stdio`

  --stdio=true: Set this to true to forward the tunnel to stdout/stdin; Only works with a single port

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v restart

Usage:

```
d8 v [command] [args...] restart (VirtualMachine) [options]
```

Restart a virtual machine.

Options

- `--all`

  Set this flag to apply the action to all VMs.
- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--label-selector`, `-l`

  Set this flag to apply the action to VMs with the specified labels.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.
- `--yes`, `-y`

  Set this flag to confirm the action without prompting for confirmation.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v scp

Usage:

```
d8 v [command] [args...] scp [flags] SOURCE TARGET [options]
```

Secure file CoPy from/to a virtual machine.

Options

- `--identity-file`, `-i`

  Specify a path to a private key passed to the local SSH/SCP client as -i; If not provided, OpenSSH default identity selection applies
- `--known-hosts`

  Set a path to the known_hosts file passed to the local SSH/SCP client as UserKnownHostsFile.
- `--local-ssh`

  Deprecated: local SSH/SCP client is always used
- `--local-ssh-opts`

  Deprecated: use --ssh-args instead
- `--port`, `-p`

  Specify a port to connect to
- `--preserve`

  Preserves modification times, access times, and modes from the original file.
- `--recursive`, `-r`

  Recursively copy entire directories
- `--ssh-args`

  Additional options to be passed to the local SSH/SCP client. May be repeated, or a single value may contain several options separated by spaces (e.g. --ssh-args='-o X -o Y').

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v spice

Usage:

```
d8 v [command] [args...] spice VirtualMachine [options]
```

Open a spice connection to a virtual machine.

Options

- `--address`

  --address=127.0.0.1: Setting this will change the listening address of the SPICE proxy. Example: --address=0.0.0.0 will make the proxy listen on all interfaces.
- `--force`

  Connect without asking, even when somebody else is using the SPICE display.
- `--port`

  --port=0: Assigning a port value to this will try to run the proxy on the given port if the port is accessible; If unassigned, the proxy will run on a random port
- `--preferred-compression`

  --preferred-compression=glz: Ask the server for this image codec: auto-glz, auto-lz, quic, glz, lz, off. Overrides what the VM is configured with, so codecs can be compared without restarting it.
- `--proxy-only`

  --proxy-only=false: Setting this true will run only the spice proxy and show the port where SPICE viewers can connect

Global options

- `--as`

Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ssh

Usage:

```
d8 v [command] [args...] ssh [-n|--namespace NAMESPACE] VIRTUAL-MACHINE-NAME [-- COMMAND [ARGS]...] [options]
```

Open a SSH connection to a virtual machine.

Options

- `--command`, `-c`

Specify a command to execute in the VM. Equivalent to passing the command after --.

- `--identity-file`, `-i`

  Specify a path to a private key passed to the local SSH/SCP client as -i; If not provided, OpenSSH default identity selection applies
- `--known-hosts`

  Set a path to the known_hosts file passed to the local SSH/SCP client as UserKnownHostsFile.
- `--local-ssh`

  Deprecated: local SSH/SCP client is always used
- `--local-ssh-opts`

  Deprecated: use --ssh-args instead
- `--port`, `-p`

  Specify a port to connect to
- `--ssh-args`

  Additional options to be passed to the local SSH/SCP client. May be repeated, or a single value may contain several options separated by spaces (e.g. --ssh-args='-o X -o Y').
- `--username`, `-l`

  Specify user to log into virtual machine; If unassigned, this will be empty and the SSH default will apply

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v start

Usage:

```
d8 v [command] [args...] start (VirtualMachine) [options]
```

Start a virtual machine.

Options

- `--all`

  Set this flag to apply the action to all VMs.
- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--label-selector`, `-l`

  Set this flag to apply the action to VMs with the specified labels.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.
- `--yes`, `-y`

  Set this flag to confirm the action without prompting for confirmation.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v stop

Usage:

```
d8 v [command] [args...] stop (VirtualMachine) [options]
```

Stop a virtual machine.

Options

- `--all`

  Set this flag to apply the action to all VMs.
- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--label-selector`, `-l`

  Set this flag to apply the action to VMs with the specified labels.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.
- `--yes`, `-y`

  Set this flag to confirm the action without prompting for confirmation.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v vnc

Usage:

```
d8 v [command] [args...] vnc VirtualMachine [options]
```

Open a vnc connection to a virtual machine.

Options

- `--address`

  --address=127.0.0.1: Setting this will change the listening address of the VNC server. Example: --address=0.0.0.0 will make the server listen on all interfaces.
- `--compress`

  --compress=6: Compression effort from 0 to 9. Higher spends more CPU on both ends; above 6 the gain is negligible
- `--force`

  Connect without asking, even when somebody else is using the VNC.
- `--port`

--port=0: Assigning a port value to this will try to run the proxy on the given port if the port is accessible; If unassigned, the proxy will run on a random port

- `--proxy-only`

  --proxy-only=false: Setting this true will run only the vnc proxy and show the port where VNC viewers can connect
- `--quality`

  --quality=6: Image quality from 0 to 9. Up to 7 the server encodes smooth areas as JPEG, 8 and 9 stay lossless at the cost of bandwidth

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Glossary

The glossary lists the terms used in the Deckhouse Platform documentation and explains what each of them means. Technical details and implementation details are described in the Architecture section.

Active-active balancing

A scheme of balancing incoming traffic in a multicluster in which the external load balancer distributes traffic across several active clusters at once, for example by geographic proximity or current load.

In the active-active scheme all clusters of the multicluster process traffic simultaneously. Traffic goes to the nearest or least loaded cluster, or follows other defined rules. Shared data is replicated between the clusters. When one cluster fails, traffic is automatically rerouted to the others. The balancing is configured by the cluster administrator as part of geo-reserving. The alternative scheme is active-standby.

For details, see Geo-reserving.

Active-standby balancing

A scheme of balancing incoming traffic in a multicluster in which the external load balancer routes all traffic to the currently active cluster, and the standby cluster receives traffic only when the active one becomes unavailable.

In the active-standby scheme the active cluster of the multicluster receives all traffic, and the standby cluster stays idle until the active one fails. Traffic failover to the standby cluster happens automatically when problems are detected. The balancing is configured by the cluster administrator as part of geo-reserving. The alternative scheme is active-active.

For details, see Geo-reserving.

Additional network

A software-defined network for pods and virtual machines on top of the main cluster network, provided by the sdn module. Additional networks are cluster networks or project networks.

A cluster network is publicly available in every project and is managed by the administrator. A project network is available within one namespace and is created by the user. Pods are connected to additional networks with a pod annotation, virtual machines with the list of networks in the machine specification. IP addresses on the additional interfaces are assigned by the IPAM of the `sdn` module.

For details, see Additional networks for use in application pods.

Additional project namespace

A user namespace created within a project in addition to the main one. It is used only to run the workload.

Additional namespaces let you divide the workload of a project logically and apply separate access policies to each part. Like the main project namespace, an additional project namespace is a user namespace.

Addon-operator queue

A task queue of the Deckhouse controller that processes built-in modules and modules from a source. Each queue is a separate pipeline with one worker that executes tasks from the head until they succeed.

The Deckhouse controller creates three kinds of queues:

- the `main` queue, which runs global hooks at startup, installs critical modules and configures or deletes modules;
- several parallel queues `parallel_queue_0..N` for module runs with dependencies taken into

account;
- hook queues named in the configuration of a hook.

A task that fails and has no `allowFailure` blocks its queue, and it is restarted with an exponential delay starting at 5 seconds. Queues do not block each other. The queues can be inspected with `d8 system queue list`.

For details, see Queuing mechanism.

Admission policy

A rule that is applied to objects such as pods and Services when they are created or modified in the cluster, based on their manifests, and not while they run.

Admission policies formalize the parameters that are allowed or prohibited in object manifests. Deckhouse Platform implements them with the `admission-policy-engine` module and divides them into three categories: Pod Security Standards, operational policies (OperationPolicy) that validate parameters not related to security, such as allowed image registries, and security policies (SecurityPolicy) that validate security-related parameters, such as privileged mode or host namespaces.

The policies complement each other: if several apply to a namespace, an object must satisfy all of them. When a policy changes, existing pods are validated on the next restart, and Deckhouse Platform raises an alert for running objects that violate a policy. SecurityPolicyException allows fine-grained exceptions for individual pods or containers.

For details, see Security policies.

Advanced load balancer

A load balancer of the `service-with-healthchecks` module that runs active probes for individual TCP ports instead of relying on container readiness probes, so that each balancer serving the same pod works independently.

The advanced load balancer is built on ServiceWithHealthchecks objects. It consists of a controller that runs on the master nodes and manages these objects, and agents on every node that run the probes and publish only verified addresses. The standard load balancer, a Kubernetes Service, remains available for workloads that need no separate probes.

For details, see Intra-cluster communication.

Air-gapped cluster

A cluster without access to an external container image registry. The only source of images for it is the in-cluster cache.

In the `Managed` registry mode the cluster is considered isolated when the `primary.upstream` parameter is not set. The in-cluster cache is then filled with the `d8 mirror push` command and serves as the registry of Deckhouse Platform components.

For details, see Managing the registry in DP-managed clusters.

ALB (application load balancing)

Load balancing at the application level. HTTP(S) headers, paths and domains are analysed, SSL termination and content-based routing are supported.

Deckhouse Platform provides three ALB implementations: Ingress NGINX Controller (the `ingress-nginx` module), Kubernetes Gateway API (the `alb` module) and Istio (the `istio` module). Balancing at the network level is NLB.

For the criteria of choosing an implementation, see Incoming traffic balancing.

Alert

A notification of the Deckhouse Platform monitoring system about an event in the cluster or its components.

The standard delivery includes a set of basic alerts that cover the state of the cluster and its components. Custom alerts can be added. Alerts are sent to external systems through Alertmanager: by SMTP, to PagerDuty, Slack, Telegram or by webhook.

The list of all alerts of the monitoring system is given in the Alerts reference.

Alert notification channel

A delivery channel through which the monitoring system of Deckhouse Platform sends alerts to recipients, such as Telegram, email or a webhook.

Notification channels, notification policies and silences are configured in the `observability` module. Alerts are delivered by Alertmanager, which in Deckhouse Platform supports multitenancy: alerts are divided into system (cluster) and project ones, and channels and delivery policies can be configured separately in every project.

For details, see Observability module.

Alert severity

The Severity (S) parameter of an alert that indicates its level of criticality, from S1 for a critical failure that requires immediate action to S4–S9 for a low-level problem that does not impair overall performance.

Every alert description contains the Severity parameter with a value from `S1` to `S9`:

- `S1` is the maximum level, a critical failure that requires immediate action;
- `S2` is a high level close to the maximum, a possible accident that requires rapid response;
- `S3` is a medium level, a potentially serious problem that requires verification;
- `S4` to `S9` are low levels, a problem that does not impair overall performance.

The severity of every alert is shown in the list of alerts.

Application endpoints

The addresses of the Ingresses of an application that Deckhouse Platform publishes in the `status.urls` field of the Application. The web interface shows them as links to the application.

To publish an endpoint, the package adds the `packages.deckhouse.io/application-endpoint-description` annotation to the Ingress: its value is the endpoint description, `"true"` adds the endpoint without a description, `"false"` excludes the Ingress. Deckhouse Platform builds a URL for every `host` and `path` pair of the Ingress rules, with the `https` scheme if the host is listed in `spec.tls`, and updates `status.urls` of the application instance after every successful apply of the templates.

For details, see Templates.

Application instance

An application installed from a Marketplace package as an Application object in a namespace. The same package can be installed several times with separate names and settings, and the instance name is a part of the names of all application objects.

Each instance is fully independent: two Redis instances, one for caching and one for sessions, can live in the same namespace. Deckhouse Platform processes every instance in a separate queue. The objects of an instance are named `d8a-<INSTANCE_NAME>-<SUFFIX>`, so the instance name is limited to 24 characters, and every rendered object carries the `packages.deckhouse.io/instance` label.

For details, see Installing and managing applications.

Application reconciliation run

The configuration, hook initialization and apply steps that Deckhouse Platform performs for an application instance. A new run starts when the settings change.

Deckhouse Platform processes each application instance in a separate queue: it downloads the package bundle, loads it, waits until the package requirements are met, validates and applies the settings, enables the hook bindings and runs the `onStartup` hooks, then runs `beforeHelm` hooks, renders the templates and applies them with Nelm, and runs `afterHelm` hooks. The last three steps form a reconciliation run. The Helm release is upgraded only when needed: on the first installation, if the rendered manifests have changed, or if some resources are missing in the cluster.

A new run starts when `spec.settings` or `spec.maintenance` of the Application changes or when a `schedule` or Kubernetes hook changes values.

For details, see Lifecycle and debugging.

Application suspension

The state of an installed application whose package requirements have stopped being met, for example a mandatory module was disabled. Deckhouse Platform uninstalls the Helm release but keeps the Application in the Suspended state.

Deckhouse Platform checks the package requirements all the time, not only on admission. When they stop being met, it runs the `beforeDeleteHelm` hooks, uninstalls the Helm release, which deletes the resources of the application instance, and runs the `afterDeleteHelm` hooks. The orphan resources are not deleted. When the requirements are met again, the application is installed again with the same settings.

The resources are deleted together with the Helm release, including PersistentVolumeClaims. To keep the data, add the `helm.sh/resource-policy: keep` annotation to them.

For details, see Lifecycle and debugging.

Arbiter node

A node that holds only etcd, without the other control plane components, to provide the etcd quorum in HA mode with two master nodes.

HA mode with two master nodes and an arbiter node meets the high availability requirements with limited resources. The arbiter node must have the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that keeps user workloads off it. It needs at least 2 CPU cores, 4 GB of RAM and 8 GB of disk space for etcd, and the same network latency as the master nodes.

For details, see Managing HA mode.

Automatic certificate issuance

Obtaining TLS certificates automatically with the `cert-manager` module, either through a Certificate object that references a ClusterIssuer or through the tls-acme annotation on an Ingress.

With a Certificate object, `cert-manager` runs the domain ownership challenge by the method defined in the ClusterIssuer, for example `HTTP-01` or `DNS-01`, creates a temporary Ingress for the challenge if needed and stores the certificate in the Secret. The `tls-acme` annotation on an Ingress requests a certificate without a separate Certificate object. Deckhouse Platform supports certificates from Let’s Encrypt, HashiCorp Vault and Venafi, self-signed certificates, automatic renewal and expiration monitoring.

For details, see Configuration of using TLS certificates.

Availability probe

A check that the upmeter module runs regularly to calculate the availability of Deckhouse Platform components. Probes are grouped, and groups or individual probes can be disabled.

The upmeter-agent component runs on master nodes and executes probe groups: control-plane (API server availability and controller health), deckhouse (cluster health and the deckhouse controller), extensions, load-balancing, monitoring-and-autoscaling, nginx (every Ingress controller has a `Ready` pod), nodegroups and synthetic (network connectivity between nodes). For example, the control-plane group creates a namespace, a pod and a StatefulSet and checks that Kubernetes processes them.

Probe results are stored by the upmeter controller excluding the downtime intervals defined in Downtime objects, shown on the status page and the availability dashboard, and used for SLA monitoring. Probes or probe groups are disabled with the `disabledProbes` parameter of the module.

For details, see Upmeter module.

Availability zone

The basic unit of a disaster-resilient architecture: a logical group of one or several data centers located within a few kilometers of each other and viewed as an isolated unit. Availability zones are part of regions.

An availability zone (AZ) is physically distributed infrastructure that can be treated as a virtual data center. The data centers of one zone are separated far enough (usually 10 km or less) to survive local incidents such as floods or fires, keep low network latency between each other, have independent power supply and use different network providers and routes. Several zones form a region.

Deckhouse Platform lets you distribute cluster nodes across availability zones (Multi-AZ) so that applications keep running when one zone of a data center or cloud becomes unavailable. With most cloud providers the zone distribution happens automatically. Outside clouds the administrator labels nodes with their zone. This is one form of geo-distribution.

For details, see Geo-distribution.

bashible

The service that runs on every cluster node and executes the bash scripts, also called bashible, that configure the node according to its node group.

The scripts are Go templates grouped into bundles. An individual script is called a step. They are written so that they can be re-run safely. The service re-runs the steps when the configuration checksum of the node group changes. The steps are stored on the node in `/var/lib/bashible/bundle_steps/`. NodeGroupConfiguration adds custom steps to the bashible configuration of a group.

bashible is part of the Cluster & Infrastructure subsystem. It runs at the operating system level as a system service and is not part of the `node-manager` module.

For details, see Bashible.

Bashible bundle

A collection of bashible scripts. Four bundles exist, bashible installation scripts, bootstrap scripts for the first node, node configuration scripts of a cloud provider and common scripts. An individual script of a bundle is a step.

bashible scripts are Go templates, so a bundle is rendered for a specific node group, and every step must be safe to run again after a failure. Rendered bundles are served by bashible-api-server. The steps cover the main node configuration stages: access for NodeUser, CA certificates, binaries in `/opt/deckhouse/bin`, packages, containerd, kubernetes-api-proxy, kubelet and the bashible service itself.

For details, see Bashible.

bashible-api-server

A Kubernetes extension API server on the master nodes that generates bashible scripts from templates stored in custom resources.

Storing every variant of the scripts for all supported operating systems in etcd is not feasible because of the key size limit and the load, so kube-apiserver forwards requests for resources with bashible bundles to bashible-api-server and returns the generated result. It serves the second phase of the bootstrap script, the `bashible.sh` script and the rendered bundle of a node group.

For details, see Bashible.

Bastion host

A host through which the master node is reached over SSH when the cluster nodes have no public IP addresses.

A bastion host is needed in layouts where nodes are created without public IP addresses. The installer connects through it when run with the `--ssh-bastion-host`, `--ssh-bastion-user` and, if needed, `--ssh-bastion-port` parameters.

Bootstrap script

The bootstrap.sh script that Deckhouse Platform generates for every NodeGroup to connect a server to the cluster as a node of that group.

The script is stored in the `manual-bootstrap-for-<NODE-GROUP>` secret in the `d8-cloud-instance-manager` namespace and is also available in the Deckhouse web interface. For manual connection of a static node, copy the script to the server and run it as `root`. CAPS runs it over SSH itself. The script installs and configures the operating system settings, containerd, kubelet and the `kubernetes-api-proxy` component, then registers the node in the cluster.

For details, see Adding and managing bare-metal nodes.

bpfLB

The BPF Load Balancer mode of the `cni-cilium` module that processes external traffic for Services of the NodePort and LoadBalancer types. The mode is set by the bpfLBMode parameter.

In the `SNAT` mode, the BPF load balancer fully replicates the logic of `kube-proxy`: the source IP address of packets sent to pods on other nodes is replaced with the InternalIP of the node. In the DSR

mode, incoming traffic passes through the load balancer and outgoing traffic goes directly from the node of the target pod, which increases the overall throughput.

For details, see BpfLB operation modes.

Built-in module

A module included in Deckhouse Platform. Its release cycle is tied to the release cycle of the platform.

Built-in modules are delivered together with a Deckhouse Platform release and are updated with it. Which of them are enabled by default depends on the module bundle and the edition. The other kind of modules are modules from a source, whose releases do not depend on the platform releases.

For details, see Deckhouse Platform module development.

Caching DNS server

A DNS server that the `node-local-dns` module runs on every cluster node. It caches DNS responses and removes the network round trip to the cluster DNS for repeated requests.

On each node the module configures an interface with the IP address of the `kube-dns` service, starts a caching CoreDNS on that address and adds an iptables rule that redirects DNS traffic to it while its socket is open. Otherwise the standard routing through the cluster DNS is used. External requests that are not cached yet are still resolved through the chain of internal zones first, but under load the cache alone improves resolution performance noticeably.

For details, see Caching DNS server in a cluster.

Canary deployment

An application deployment strategy that rolls out a new version to a small portion of traffic first and shifts more traffic to it as confidence grows, with a quick rollback to the old version if issues arise.

In Deckhouse Platform a canary deployment is implemented with the `ingress-nginx` module through Ingress annotations or with the `istio` module through VirtualService and DestinationRule objects (the Istio way is recommended). The new version runs as a separate Deployment and Service next to the stable one.

For details, see Configuring Canary deployment.

Chaos engineering

Deliberate or random disruption of cluster components to test its resilience. In Deckhouse Platform it is enabled per node group with the spec.chaos parameter, only for CloudEphemeral nodes.

The `spec.chaos` parameter of a node group of CloudEphemeral nodes sets the mode and the period. In `DrainAndDelete` mode Chaos Monkey drains and deletes a node of the group once per `period`. The `Disabled` mode skips the group. The mode can also be enabled in the web interface in the node group settings.

For details, see Chaos engineering.

CIS compliance check

A check that Deckhouse Platform runs automatically in every cluster against the CIS Kubernetes Benchmark recommendations. The results are available as reports and on the Security / CIS Kubernetes Benchmark dashboard.

Deckhouse Platform follows the CIS Kubernetes Benchmark both at the level of individual components and for the platform as a whole. Access to the reports, like access to vulnerability scanning results, is granted through the role model of the `operator-trivy` module.

For details, see Configuration of container image vulnerability scanning.

Cloud cluster

A cluster with the clusterType: Cloud. Deckhouse Platform has access to the API of the cloud infrastructure and creates and manages the virtual machines of the nodes.

The installer creates all the required resources, such as virtual machines and networks, from the `<PROVIDER>ClusterConfiguration` of the cloud provider. The master nodes of a cloud cluster are CloudPermanent nodes, worker nodes are usually CloudEphemeral nodes.

For details, see Platform installation.

Cloud provider

A cloud platform or virtualization system that Deckhouse Platform integrates with through a cloud-provider-* module to create virtual machines, connect them to the cluster as nodes and create base infrastructure resources.

The workflow is the same for all providers, only the preparation steps, such as creating a service account, and the configuration files differ. The integration enables CloudEphemeral nodes, which Deckhouse Platform creates and deletes through the provider API, and the choice of a layout.

For details, see Integration with IaaS.

CloudEphemeral node

A temporary cloud node that Deckhouse Platform orders, creates and deletes automatically through the cloud provider API. It is described by a NodeGroup with the CloudEphemeral node type.

The parameters of the virtual machines are taken from an instance class, the number of nodes and their zones from the `cloudInstances` section of the node group. The nodes are created by Machine Controller Manager or Cluster API, which are part of the `node-manager` module.

CloudEphemeral groups support node group autoscaling, standby nodes and chaos engineering.

For details, see Adding and managing cloud nodes.

CloudPermanent node

A permanent cloud node whose configuration is taken from the provider cluster configuration rather than from a NodeGroup. The master nodes of a cloud cluster are CloudPermanent nodes.

CloudPermanent nodes are described in the `nodeGroups` section of the `<PROVIDER>ClusterConfiguration` structure, and their number is set by `replicas`. The nodes are managed by Terraform built into Deckhouse Platform, and configuration changes are applied by running the installer with `dhctl converge`.

For details, see Adding and managing cloud nodes.

CloudStatic node

A node created manually or by external tools in the cloud for which the provider integration is configured, and connected to the cluster with the bootstrap script. CSI runs on it, and cloud-controller-manager manages its Node object.

The Node object of a CloudStatic node automatically receives the zone and region from the provider data. If the machine is deleted from the cloud, its Node object is deleted from the cluster. Like a static node, a CloudStatic node does not support automatic scaling and updates.

For details, see Adding and managing cloud nodes.

Cluster

A group of interconnected nodes managed by Deckhouse Platform that provides a single environment for deploying, scaling and operating workloads.

A cluster consists of the system, which keeps the cluster running, and the workload, which runs in the cluster.

Cluster API Provider Static (CAPS)

The mechanism that connects, configures and, when needed, cleans up static nodes automatically, based on StaticInstance and SSHCredentials objects.

CAPS is enabled by the `staticInstances` section of a node group with the Static type: `count` sets how many nodes the group must have, `labelSelector` selects the StaticInstance objects to use. CAPS keeps the number of nodes equal to `count`: it connects to a server over SSH, runs the bootstrap script and adds the node to the cluster, and when a node is removed it cleans the server up and returns its StaticInstance to the `Pending` state for reuse.

For details, see Adding and managing bare-metal nodes.

Cluster configuration (ClusterConfiguration)

The structure that holds the general parameters of the cluster, such as the Kubernetes version, network settings, CRI parameters and the cluster type.

The cluster configuration is defined at installation and edited in a running cluster with the `d8 system edit cluster-configuration` command. After the change is saved, Deckhouse Platform reconciles the cluster state with the new configuration. Some of its parameters are protected and cannot be changed in a running cluster by default.

The `clusterType` parameter defines whether the cluster is static or cloud.

For details, see Deckhouse Platform configuration.

Cluster managed by Deckhouse Platform

A cluster in which Deckhouse Platform manages both the control plane and the nodes. The opposite is a Managed Kubernetes cluster, where the control plane is managed by a cloud provider.

Deckhouse Platform can also be installed into an existing Managed Kubernetes cluster, such as GKE, EKS or AKS. In this case Deckhouse Platform does not manage the control plane, and some features are limited by the API of the cloud provider: for example, node management and the `registry` module are available only in clusters fully managed by Deckhouse Platform.

For details, see Platform installation.

Cluster network

An additional software-defined network that is publicly available in every project of the cluster and is configured and managed by the administrator.

Examples of a cluster network are a public WAN network or a shared network for traffic exchange between projects. The administrator creates it with a ClusterNetwork object of the `sdn` module and, if needed, enables IPAM for it with a cluster-wide address pool. To use such a network in application pods, contact the cluster administrator. The other kind of additional network is the project network.

For details, see Software-defined networking (SDN).

ColdStandby

A mechanism that returns a virtual machine to service on another node after the failure of the node it was running on.

The mechanism requires the `runPolicy` of the virtual machine to be `AlwaysOnUnlessStoppedManually` or `AlwaysOn`, and Fencing to be enabled on the nodes that run virtual machines. Without Fencing, an unavailable machine does not move: it stays on the failed node and resumes together with it.

For details, see Virtual machine fault tolerance and balancing.

Conditional module dependency

A dependency on another module that applies only if that module is enabled. The module works on its own, but if the other module is present its version must match the constraint.

A conditional dependency is a kind of module dependency and is available in Deckhouse Platform starting with version 1.73. It may affect the ability to enable, disable and update both modules: the dependent one and the one it depends on. A Marketplace package lists such modules in `requirements.modules.conditional`.

For details, see Dependencies of the Deckhouse Platform modules.

Consistent snapshot

A snapshot of a virtual machine disk that captures a coherent and integral state of its data. It is created when the disk is not attached to any machine, the machine is powered off, or the guest OS agent freezes the file system while the snapshot is taken.

Whether a consistent snapshot is required is set by the `requiredConsistency` parameter, which is `true` by default. With a running machine the file system is frozen by the guest OS agent. Its hook scripts can also prepare applications for the snapshot without stopping services.

An inconsistent snapshot is created when the machine is running and the agent is not installed or not running. Recovery from such a snapshot can lead to file system integrity problems.

For details, see Snapshots of disks and virtual machines.

Control plane

The Kubernetes components that manage the cluster, such as kube-apiserver and etcd. In Deckhouse Platform they run on master nodes, and in HA mode with two master nodes the etcd quorum is provided by an arbiter node.

Deckhouse Platform manages the certificates of the control plane, generates the configuration and manifests of its components, upgrades and downgrades their versions, manages the etcd cluster and its members, runs etcd defragmentation and maintains the kubeconfig files. The `control-plane-manager` module runs on all master nodes.

The control plane works in the single-master or multi-master mode. In HA mode with two master nodes, the third etcd member runs on an arbiter node: it holds only etcd, without the other control plane components, so that the etcd cluster keeps its quorum.

For details, see General management and configuration of the control plane.

Core fraction

The guaranteed share of the power of each virtual core of a virtual machine, set by the coreFraction parameter. Without it every virtual core gets 100% of a physical one.

A machine with `cores: 2` and `coreFraction: 20%` is guaranteed 0.4 cores in total regardless of the node load and can take both cores in full when the node has free resources. Such a reserve keeps more machines on a node than it has physical cores. This is how CPU oversubscription works. The guaranteed share is taken into account when a node is selected, so a machine does not start where the node cannot provide the guarantees to all machines placed on it.

An administrator can restrict the allowed values in the sizing policy of the VM class. In commercial editions `coreFraction: Auto` lets Deckhouse Platform pick the share by the actual consumption of the machine.

For details, see Virtual machine CPU and memory.

CPU oversubscription

Giving the virtual machines on a node more virtual cores than the node physically has. The degree of oversubscription is controlled by the core fraction guaranteed to a virtual machine.

The `coreFraction` parameter of a virtual machine defines the share of a core’s capacity guaranteed to it. The allowed values are set in the sizing policy of the class. With `coreFraction: 20%`, a machine always gets a fifth of a core and can take a whole core when the node has spare resources. When placing a machine, Deckhouse Platform sums the guaranteed shares of all machines on the node, and if the sum exceeds the number of physical cores, the machine does not start on that node.

For details, see CPU oversubscription for virtual machines.

Critical module

A module marked with critical set to true in module.yaml. It starts during the initial cluster bootstrap, before the cluster is considered fully ready. Other modules start only after that.

The Deckhouse controller puts the tasks of critical modules into the `main` queue in ascending order of module weight. Non-critical modules are processed in the parallel queues without regard to weight. If critical modules can be processed in parallel, the controller runs them as a batch and waits for completion before continuing the `main` queue.

For details, see Queuing mechanism.

CSI driver

A plugin that manages persistent storage volumes in Deckhouse Platform. It is part of the cloud-provider-* and csi-* modules, and all implementations share one reference architecture.

Container Storage Interface (CSI) is a standard interface that unifies access to storage systems. Every cloud provider and external storage system has its own implementation of the specification. The implementations differ in the supported capabilities and components, but not in architecture.

For details, see CSI driver.

Dashboard

A collection of charts and tables with data about the state of the cluster and the applications running in

it. Deckhouse Platform ships preinstalled dashboards and lets users create custom ones.

Dashboards show CPU usage, memory consumption, disk and network activity, and the status of pods, controllers, nodes and namespaces. They are available in the Deckhouse web UI under “Monitoring” → “Dashboards”.

Preinstalled dashboards are shipped with Deckhouse Platform and cannot be edited. Custom dashboards are created with an ObservabilityDashboard object of the `observability` module at the namespace level with access control, or with a GrafanaDashboardDefinition object at the cluster level, which is a legacy approach.

For details, see Monitoring dashboards.

Deckhouse CLI (d8)

The command-line interface for working with the products of the Deckhouse ecosystem. It combines commands for Kubernetes, Deckhouse Platform administration, application delivery, image copying, backups, diagnostics, virtualization and other tasks.

Starting from release 1.59, `d8` is installed on all Deckhouse Platform cluster nodes. It can also be installed on an administrator machine with trdl (an open-source tool for securely delivering and updating binary files) or from the executable file. Commands are grouped by purpose: `d8 k` is equivalent to `kubectl`, `d8 system` works with platform settings, modules, queues and packages, `d8 mirror` copies platform images, `d8 v` manages virtual machines, `d8 backup` creates backups.

For details, see Deckhouse CLI overview and installation.

Deckhouse controller

The controller of the deckhouse module that implements platform management operations and coordinates them through the queueing mechanism. The deckhouse module is the only module without which Deckhouse Platform cannot run.

The Deckhouse controller manages modules: it runs their hooks, applies module settings and installs module releases. Tasks are processed in the addon-operator queues. In HA mode several controller instances run and elect a leader through the `deckhouse-leader-election` Lease. The leader performs all management operations.

The controller can run in standard mode or in hook isolation mode, in which shell hooks and module enable scripts run in a chroot environment with a limited set of mounted directories.

For details, see Deckhouse module.

Deckhouse Platform API

The interface through which users and Deckhouse Platform components interact with the cluster. It is fully compatible with the Kubernetes API.

All operations in Deckhouse Platform are performed through the API, which keeps the interaction uniform. The API of every module, its custom resources and parameters, are described in the module documentation.

Deckhouse Platform component registry

The container image registry from which the cluster pulls the images of Deckhouse Platform components.

The registry is specified during cluster bootstrap in the `deckhouse` parameter of InitConfiguration. How its configuration is managed in a running cluster depends on how the cluster is managed: in a cluster fully managed by Deckhouse Platform, the `registry` module is used and the registry management mode is set in its ModuleConfig. In a Managed Kubernetes cluster, the `helper change-registry` is used.

For details, see DP components registry.

Deckhouse Platform image bundle

An archive of Deckhouse Platform container images produced by d8 mirror pull and loaded into a private container registry with d8 mirror push. It is the way to deliver platform images into a private environment.

The bundle is downloaded from the public Deckhouse Platform registry with a license key, usually on the bastion host, and brought into the perimeter by any permitted means, for example on removable

media. Its size after `d8 mirror push` guides the disk capacity of the private registry. The `d8 mirror` commands are not available in the Community Edition and Basic Edition editions.

For details, see Installing DP in a private environment.

Deckhouse Platform release

Any published version of Deckhouse Platform. Releases are distributed through release channels with defined delays and are of two types, a patch version and a minor version.

A patch version includes bug fixes, a minor version includes new features. Each new release first appears on the Alpha release channel and then moves to the more stable channels. In the cluster a release is represented by a DeckhouseRelease object, which is applied according to the update mode and update windows. Minor releases cannot be skipped, because they contain migrations that must be applied in order.

For details, see Updating.

Deckhouse web UI

The web interface of Deckhouse Platform for cluster management, including monitoring, parameter configuration, node and module management, security and network configuration.

Most operations available from the command line through Deckhouse CLI or `kubectl` can also be performed in the web UI. It opens at `console.<CLUSTER_NAME_TEMPLATE>`, where the template is the DNS name template of the cluster, and contains the sections Deckhouse, node management, multitenancy, access, network, storage, security, monitoring and logging.

For details, see Deckhouse Platform web UI.

Deletion confirmation

A confirmation that the user must give in the web interface before an application is deleted or a module is disabled. It is declared in the disable section of package.yaml or module.yaml together with the message shown to the user.

For a Marketplace package the `disable` section sets `confirmation: true` and localized messages, for example a warning that deleting the application deletes the data stored in its volumes. Deckhouse Platform publishes the section in the `status.packageMetadata.disableOptions` field of the package version, and the Deckhouse web UI shows the message in the language of the user. The confirmation is not checked when an Application is deleted through the Kubernetes API.

For a module the `disable.confirmation` and `disable.message` parameters of `module.yaml` work the same way when the module is disabled.

For details, see Application development.

Deployment option

A cluster configuration that defines the number of nodes, the distribution of roles between them, and whether the cluster tolerates a node failure.

The deployment option is chosen before the installation. Examples of options: a single-node cluster, a single-master cluster, a high-availability cluster with three master nodes, and a cluster with role separation, where system, frontend and monitoring nodes are dedicated to the platform components and incoming traffic. The node resource requirements depend on the deployment option.

For details, see the Deployment options section of the “System requirements” page.

Device passthrough

Giving a virtual machine a device of the node it runs on, such as a USB device, a GPU or a PCI device.

Each device type has its own node requirements and its own preparation order.

For details, see Virtualization.

Disaster resilience

The ability of infrastructure based on Deckhouse Platform to remain operational in the event of large-scale failures. It is achieved through distributed deployment, automatic traffic switching and replication of critical components.

Deckhouse Platform implements two approaches to disaster resilience. Geo-distribution spreads the components of one cluster across several availability zones or regions. Geo-reserving combines several independent clusters into a multicluster and redirects traffic to another cluster when one becomes unavailable.

Both approaches require network connectivity between nodes and regions, traffic balancing and a suitable storage organization. The choice of an architecture depends on the applications and the constraints of the infrastructure. Within one cluster, the redundancy of critical components is provided by the HA mode.

For details, see Disaster resilience.

Disruptive update

A node update that requires node downtime and may briefly interrupt system components, for example upgrading containerd or kubelet across several versions. The approval mode is set by the disruptions.approvalMode parameter of the NodeGroup.

Regular node updates are always applied automatically and cause no downtime or reboot. The `disruptions` section governs only disruptive updates, such as a kernel or containerd update or a significant kubelet version change. Only one node per group is updated at a time.

Three approval modes are available in a node group. In `Manual` mode the `NodeRequiresDisruptionApprovalForUpdate` alert is raised, and the update starts after the `update.node.deckhouse.io/disruption-approved=` annotation is added to the node. The node is not drained automatically. In `Automatic` mode the node is drained and updated without confirmation. In `RollingUpdate` mode a new node with the updated settings is created and the old one is removed. This mode applies to cloud nodes only.

Use `Manual` mode for the groups of master nodes.

For details, see Node management basics.

DNS name template

The cluster DNS name template from the modules.publicDomainTemplate parameter of the global ModuleConfig. Deckhouse Platform modules substitute a reserved name for the `%s` placeholder to build the DNS names of their service web interfaces.

If the template is not set, the modules do not create Ingress resources. The domain in the template must not match or be a subdomain of the cluster domain from `clusterDomain`. With a wildcard DNS record one record covers all names. With a non-wildcard template, for example `%s-kube.company.my`, an A or CNAME record is needed for each reserved name, pointing to the public IP address of the Ingress controller. The template is a part of the global settings.

For details, see List of reserved Deckhouse Platform DNS names.

DSR (Direct Server Return)

A traffic processing method in which incoming packets pass through the load balancer and outgoing packets go directly from the servers that handle the request.

Responses are usually much larger than requests, so DSR unloads the balancer and increases the overall throughput. In Deckhouse Platform, DSR is one of the bpfLB modes of the `cni-cilium` module and is used instead of SNAT.

For details, see BpfLB operation modes.

DVCR

The internal container image storage in which Deckhouse Platform keeps virtual machine images. It resides on a persistent volume of the cluster.

Images travel from DVCR to virtual machine disks, so the size of the volume determines how many images fit into the cluster. The volume size and storage class are set in the `dvcr.storage` block of the `virtualization` ModuleConfig. After the volume is created, its size cannot be reduced and its storage class cannot be changed. Stale data of deleted images and disks is removed by garbage collection on a schedule, daily at 02:00 by default.

For details, see Virtual machine image storage.

Edition

A Deckhouse Platform distribution, such as Open, Core, Ultimate, or CSE Pro, that defines the set of available modules and features.

The edition is selected when the cluster is installed. A running cluster can be switched to another edition.

For details, see Switching between DP editions.

Egress gateway

Centralized management and control of outgoing traffic from the cluster, including encryption, routing and monitoring.

The egress gateway is provided by the `cni-cilium` module and is configured with two custom resources. EgressGateway describes a group of nodes, one of which is selected as the active gateway while the others stay in standby mode. EgressGatewayPolicy describes which traffic of cluster pods is routed through that gateway.

Two modes are available: the basic mode with pre-configured IP addresses on the egress nodes and the Virtual IP mode, in which additional IP addresses are assigned to nodes dynamically. For details, see Egress Gateway configuration.

End-to-end authentication

Authentication implemented in Deckhouse Platform so that a user is verified when accessing any interface of the platform or any cluster resource. Users can also enable it for their own applications.

Depending on the configuration, authentication uses either the internal user database (static users) or external authentication providers such as LDAP, GitLab or GitHub. An external provider lets users keep their existing credentials and use them in several Deckhouse Platform clusters. For a cluster user or an application developer the interface and the authentication methods for applications stay the same whichever way the administrator configured authentication.

For details, see Configuring authentication for applications.

Extended monitoring

A monitoring mode of the `extended-monitoring` module that adds monitoring of secrets and TLS certificate expiration, collection of cluster events as metrics, checks of image availability in the registry and monitoring of objects in labelled namespaces.

Objects are monitored in the namespaces that carry the label `extended-monitoring.deckhouse.io/enabled=""`. Image availability is checked for the images used by Deployments, StatefulSets, DaemonSets and CronJobs.

For details, see Monitoring in Deckhouse Platform.

External storage

Storage systems outside the cluster nodes, Ceph, NFS, S3, SCSI and vendor storage systems, connected to Deckhouse Platform through csi-* modules, as opposed to software-defined storage.

Deckhouse Platform ships ready-to-use integrations with distributed Ceph storage, HPE, Huawei and TATLIN.UNIFIED (Yadro) storage systems, NFS, S3 object storage and SCSI-based storage. Each integration is a module with a CSI driver. The second group of storage types is software-defined storage on the disks of cluster nodes.

For details, see Storage.

Federation

An inter-cluster communication model that connects independent clusters. Each cluster keeps its own namespaces and services, and access between clusters is configured explicitly.

In a federation, the administrator defines which services are accessible from outside the cluster. The model is implemented with Istio through the `istio` module and is built on the service mesh. The model for tightly connected clusters is multicluster.

For details, see Inter-cluster cooperation.

Fencing

A mechanism that isolates a node that has lost connectivity with the cluster. The node is shut down

and, if needed, its Node object is deleted, so that no workload keeps running on it in an undefined state.

Fencing works in two stages: the node is first marked as suspicious at the gossip protocol level, then isolated. It is enabled per node group with the `spec.fencing.mode` parameter of the NodeGroup. Fencing is required for ColdStandby: without it an unavailable virtual machine stays on the failed node and resumes together with it.

For details, see Virtual machine reliability.

Frontend node

A node that balances incoming traffic. Ingress controllers run on frontend nodes.

A cluster needs more than one frontend node, and the nodes must handle the traffic when at least one of them fails. With two frontend nodes each must carry the full load, with three each must withstand a load increase of at least 150%. Frontend nodes usually form the `front` node group and run the Ingress controllers.

For details, see Recommendations for configuring cluster nodes.

Geo-distribution

An approach to disaster resilience in which the components of a cluster are distributed across several availability zones (Multi-AZ) or regions (Multi-Region) so that the cluster keeps running when a zone or region fails.

Geo-distribution reduces the impact of infrastructure failures on application availability. Distributing nodes across availability zones protects applications from the failure of one zone in a data center or cloud. Distributing them across regions lets applications survive the outage of a whole data center or cloud provider, provided that the application can work independently in each region.

The platform administrator is responsible for the network connectivity between cluster nodes, for the balancing of incoming traffic by the cloud provider or an internal load balancer, and for the organization of storage. Geo-distribution is one of the two approaches to disaster resilience in Deckhouse Platform. The other is geo-reserving.

For details, see Geo-distribution.

Geo-reserving

A method of increasing fault tolerance in which several independent Kubernetes clusters in different geographic locations are combined into a multicluster, and traffic is automatically redirected to a cluster that has an available copy of the application.

Geo-reserving requires one Deckhouse Platform cluster per region, connected with a declarative API into a multicluster: either through the built-in service mesh based on Istio or through the networking features of the platform based on Cilium. The application must be able to run in parallel in different regions, and a stable network channel must exist between the regions. A properly configured system keeps the application running even when a whole region or cloud becomes unavailable.

Incoming traffic in a multicluster is balanced by an external load balancer in one of two schemes, active-active or active-standby. The balancing and the storage organization are configured by the cluster administrator. Geo-reserving is one of the two approaches to disaster resilience. The other is geo-distribution.

For details, see Geo-reserving.

Global settings

Deckhouse Platform settings that apply to the whole cluster. They are stored in the ModuleConfig named global.

Global settings can be thought of as a special `global` module that cannot be disabled. The settings of other modules are stored in separate ModuleConfig objects named after the module.

For details, see Global settings.

Grantable cluster-wide resource

A cluster-wide resource, for example a StorageClass, that the `multitenancy-manager` module makes available to a project through a grant. Application settings bound to such a resource get the project default and are validated against the resources available to the project.

A `settings` field of type `string` is bound to a grantable resource with the `x-deckhouse-grantable-resource` extension, which names the resource available to the project, for example `storageclasses`. When the user leaves the field empty, the resource name configured as the project default is injected into the values. A value that is not in the list of resources available to the project is rejected. The group, version and kind of the resource are defined by the grant.

For details, see Application settings.

Guest OS agent

QEMU Guest Agent installed in the guest system so that Deckhouse Platform can interact with the operating system inside the virtual machine.

The agent is needed for three things: it makes consistent snapshots of disks and machines possible, it reports information about the running system to the `.status.guestOSInfo` block, and it shows that the operating system has actually booted rather than just the virtual machine having started. The `AgentReady` condition of a running machine means that the agent is working.

Deckhouse Platform works with `qemu-guest-agent` version 5.2.0 and later. The agent can be installed and enabled by the VM provisioning script.

For details, see Initial provisioning and the guest OS agent.

HA mode

A mode in which critical Deckhouse Platform components run with redundancy, so that the failure of a single instance does not stop the service.

HA mode is enabled automatically when the cluster has more than one master node, both at the initial deployment and when nodes are added later. It can also be managed manually.

For details, see High reliability and availability.

High-level role

A role of the basic role model of Deckhouse Platform, provided by the `user-authz` module on top of Kubernetes RBAC. User, PrivilegedUser, Editor, Admin, ClusterEditor, ClusterAdmin and SuperAdmin.

High-level roles belong to the basic role model. The `user-authz` module creates them as a set of ClusterRole objects and grants them to users and groups with ClusterAuthorizationRule and AuthorizationRule. Each role extends the previous one: `User` can view pods, logs and Deployments, `Admin` manages RBAC in a namespace, `ClusterAdmin` has full access to cluster-wide resources.

The recommended granular role model uses use roles and manage roles instead of high-level roles. The models are not compatible, automatic conversion is impossible, but they can be used at the same time: the permissions of both models are summed up.

For details, see Basic authorization model.

Hook binding

An event on which a hook runs, specified in the hook configuration. A hook can run on Kubernetes events, on schedule, at startup, before or after the Helm templates are applied.

Bindings of a module hook are `onStartup`, `beforeHelm`, `afterHelm`, `kubernetes` and `schedule`. Global hooks also have `beforeAll` and `afterAll`. The hooks of a Marketplace package use the same kinds of bindings, with the `Queue` field that sets a separate queue for the `Schedule` and `Kubernetes` bindings. Hooks with the same binding run in the order of their `Order` value.

For details, see Hooks.

Host firewall

A Cilium mode in which network policies apply to the cluster nodes themselves rather than to pods.

A host firewall is configured with CiliumClusterwideNetworkPolicy and the `nodeSelector` field. Such a policy applies to the host endpoint with the label `reserved:host` and filters the traffic entering and leaving the node, including `hostNetwork` pods. It is available only in clusters with the `cni-cilium` module.

Host policies add a filter layer inside the cluster and do not replace infrastructure-level protection such as physical firewalls or cloud security groups. A mistake in them can break SSH access, the control plane, kubelet or etcd, so they are rolled out through the audit mode first. For details, see Host firewall on nodes.

Hybrid cluster

A static Deckhouse Platform cluster extended with nodes placed in the infrastructure of a supported cloud provider.

The control plane and the initial nodes remain part of the static cluster. Additional nodes are added through the cloud provider module: either created automatically through the provider API as CloudEphemeral nodes, or created in advance and connected with the bootstrap script as CloudStatic nodes.

A hybrid cluster requires L3 connectivity between the data center and the cloud.

For details, see Hybrid clusters.

Hybrid node group

A node group that includes both Static nodes deployed in the cloud and servers in the on-premises data centre, bare metal or virtual machines.

In a hybrid node group the primary workload can run on bare-metal servers while cloud instances serve as a scalable addition during peak loads. All nodes are static nodes, so the `node-manager` module works as for static nodes. The only requirement is L3 connectivity between the data centre and the cloud. A hybrid node group differs from a hybrid cluster, which combines static nodes with CloudEphemeral nodes.

For details, see Hybrid node groups and clusters.

Image signature verification

Verification of the integrity and authenticity of user application images during Kubernetes API request validation, before a pod is created, against Cosign signatures and the public keys listed in a SecurityPolicy.

At the build stage the image is signed with a private key using Cosign. The signature is published in the registry as a separate tag and does not change the image. In the cluster a SecurityPolicy with the `verifyImageSignatures` parameter lists the public keys, and the `admission-policy-engine` module with Ratify verifies the signature of every image when the pod is created. This is the part of integrity control available in all editions. Verification at the container runtime level exists only in Certified Core and Certified Pro.

For details, see Integrity control.

In-cluster image cache

A container image registry deployed on the master nodes of the cluster, from which all nodes pull the images of Deckhouse Platform components.

The cache is enabled by the `storage.cache` parameter of the `registry` ModuleConfig in the `Managed` registry management mode. While the cache is being filled, the upstream registry stays a fallback path. In an isolated cluster, the cache is the only source of images and is filled with the `d8 mirror push` command.

Infrastructure node

A collective name for the master, frontend, system and monitoring nodes of a cluster, as opposed to worker nodes that run user applications.

Infrastructure nodes run the cluster control plane and Deckhouse Platform components. The recommended resource minimums depend on the role: master nodes and monitoring nodes need fast disks, frontend nodes need the least resources, and the size of system nodes depends on whether the cluster has dedicated monitoring nodes.

For details, see Going to production.

Ingress controller

A controller that routes user requests to applications and services by the rules defined in Ingress resources.

In Deckhouse Platform, the Ingress controller is Ingress NGINX Controller, managed by the `ingress-nginx` module through IngressNginxController resources. Several Ingress NGINX Controllers can run in one cluster at the same time, one of them is the primary one. Kubernetes Gateway API (the `alb` module) and Istio Ingress Gateway also balance traffic at the application level, but they work not with

Ingress resources but with Gateway API routes and Gateway and VirtualService resources. All three solutions are implementations of ALB.

Ingress controllers are usually placed on frontend nodes. The way traffic reaches a controller is defined by its inlet. For details, see Incoming traffic balancing.

Inlet

The way incoming traffic reaches an application-level load balancer — Ingress NGINX Controller or a gateway of the `alb` module. It is set in the inlet field of IngressNginxController, ClusterALBInstance and ALBInstance resources.

For the Ingress controller, the inlet takes one of the values `LoadBalancer`, `LoadBalancerWithProxyProtocol`, `HostPort` or `HostWithFailover`. The `LoadBalancer` and `LoadBalancerWithProxyProtocol` inlets use a load balancer of the cloud provider or the built-in `metallb` module. The `HostPort` inlet exposes the controller on node ports, for example behind an external load balancer. The `HostWithFailover` inlet is used on bare metal, where the controller runs on frontend nodes.

For gateways of the `alb` module, the inlet takes one of the values `LoadBalancer`, `HostPort` or `ClusterIP`. ClusterALBInstance supports `LoadBalancer` and `HostPort`, ALBInstance supports `LoadBalancer` and `ClusterIP`. The `ClusterIP` inlet accepts traffic through a service of the `ClusterIP` type, without an external entry point.

For examples of each inlet, see ALB with Ingress NGINX Controller and the Inlets section of the “ALB with Kubernetes Gateway API” page.

Installation configuration file

A set of YAML documents with Deckhouse Platform settings and manifests of the cluster objects to be created after installation. It is passed to the CLI installer with the –config parameter.

The file contains InitConfiguration (required), ClusterConfiguration (required unless Deckhouse Platform is installed into an existing cluster), StaticClusterConfiguration or the provider cluster configuration, and optionally ModuleConfig, NodeGroup and other objects. The CLI installer reads it during preflight checks and the cluster bootstrap.

For details, see Installation.

Installation into an existing cluster

An installation mode in which Deckhouse Platform runs on top of an existing Kubernetes cluster and installs and updates platform modules, but does not manage the operating system, control plane or node lifecycle.

In this mode the existing infrastructure keeps managing the operating system, kubelet, containerd, etcd, the control plane and Kubernetes updates, the existing CNI provides pod networking, and machines are created and deleted by an external provisioner or the user. The module bundle `Managed` is selected during installation and cannot be changed afterwards. ClusterConfiguration is not required in the installation configuration file.

For details, see Installing Deckhouse Platform in an existing Talos cluster.

Installer

The tool that deploys a Kubernetes cluster and installs Deckhouse Platform in it. It is available as the `dhctl` CLI installer in a container image and as the graphical installer.

The `dhctl` CLI installer deploys a cluster in a supported cloud, on bare-metal servers, including hybrid clusters, or installs Deckhouse Platform into an existing Kubernetes cluster. In a cloud cluster it creates the required infrastructure: virtual machines, networks and other resources. In a running cluster the `dhctl converge` command applies configuration changes of CloudPermanent nodes and changes of protected parameters.

The second form is the graphical installer available on the Deckhouse Platform website.

For details, see Platform installation.

Instance class

A provider-specific resource, such as OpenStackInstanceClass or AWSInstanceClass, that describes the parameters of the virtual machines Deckhouse Platform creates for a node group.

A node group of CloudEphemeral nodes refers to an instance class in `cloudInstances.classReference`. The instance class defines the machine type, the image, disks and other provider-specific parameters, while the number of nodes and their zones are defined by the node group.

For details, see Adding and managing cloud nodes.

Integrity control

A set of mechanisms that verify containers to ensure their security and compliance with the specified configuration, including image signature verification at startup, runtime audit of running containers and immutable images of platform modules.

Deckhouse Platform provides three mechanisms: image signature verification of user workloads during Kubernetes API request validation, runtime audit of running containers, and integrity protection of Deckhouse Platform modules. Cryptographic verification of image signatures at the container runtime (CRI) level with DM-Verity protection of unpacked layers is available only in the certified editions.

Starting with version 1.74, modules are installed as read-only EROFS images mounted through DM-Verity, which protects an installed module from being replaced or modified on the node. This mechanism is enabled automatically when the node OS supports the `erofs` kernel module. Without it Deckhouse Platform keeps working and raises an alert.

For details, see Integrity control.

IPAM

An IP address management mechanism that allocates addresses to the network interfaces of pods and virtual machines and ensures that no address is assigned twice. In Deckhouse Platform the CNI modules, the sdn module and the virtualization module each have their own IPAM.

The `cni-cilium`, `cni-flannel` and `cni-simple-bridge` modules assign pod addresses from the pod address range allocated to the node. This IPAM is not configurable.

In the `sdn` module, IPAM allocates IPv4 addresses from pools to the additional network interfaces of pods and virtual machines. It is enabled for each network separately by an address pool reference in Network or ClusterNetwork. For cluster networks the pools are created by the administrator, for project networks by users. If a pod is connected to multiple networks that use IPAM, specify the address objects for each interface explicitly.

The addresses of virtual machines in the main cluster network are managed by default by the `virtualization` module through the VirtualMachineIPAddress and VirtualMachineIPAddressLease. When a pool is configured for the main network in the `sdn` module, the addresses of virtual machines are managed by the shared IPAM of that module: the address is requested automatically, and existing virtual machines are brought under its management without changing their addresses or restarting.

For details, see Additional networks for use in application pods and IP addresses of virtual machines.

Istio Ingress Gateway

The entry point of an application load balancer built on Istio. The administrator deploys it with IngressIstioController, and application developers publish applications with Gateway and VirtualService objects.

Publishing with Istio has two layers: the cluster administrator deploys the IngressIstioController and the related infrastructure, and developers create Gateway and VirtualService objects in their namespace. This is one of the ALB implementations in Deckhouse Platform, next to Ingress NGINX and the Gateway API with its managed Gateway.

For details, see Publishing applications using Istio.

Kubernetes API audit

Logging of requests to the Kubernetes API server according to an audit policy. By default the log is written to the `/var/log/kube-audit/audit.log` file on master nodes.

Kubernetes API audit tracks requests to the API server and helps to investigate unexpected behaviour and to meet security requirements. Deckhouse Platform configures a basic audit policy by default: it skips frequent service updates such as Endpoints, Events and leader election and logs changes of pods, nodes, ServiceAccount objects and operations in system namespaces. The policy can be extended with custom rules.

The audit log is a source for runtime audit and for security events.

For details, see Kubernetes API event audit.

Kubernetes conformance

Testing of Deckhouse Platform against the conformance test suite of the CNCF Certified Kubernetes Conformance Program. The tests are run for every supported Kubernetes minor version.

The tests are run with Sonobuoy in the `certified-conformance` mode, and the JUnit reports are published in the reference. The results are valid only when the `multitenancy-manager` module is disabled.

For details, see Kubernetes conformance test results.

kubernetes-api-proxy

An NGINX-based proxy on every cluster node that serves requests to the Kubernetes API sent to localhost and forwards them to the master nodes, providing high availability and load balancing of the API.

bashible installs and configures the proxy and keeps its list of upstream master nodes up to date. On master nodes it runs as a static pod: by default it proxies requests to the local kube-apiserver and, if that is unavailable, queries the remaining instances in turn. Requests from kubelet to `localhost` go through it.

For details, see Control plane management.

Layout

The scheme of node placement in the infrastructure of a cloud provider. It affects networking, availability of public IP addresses, outgoing traffic routing and the way nodes are accessed.

The layout is set in the `layout` parameter of the ModuleConfig of the cloud provider module. The available layouts depend on the provider, for example `Standard`, `WithoutNAT` and `WithNATInstance` in Yandex Cloud.

In some layouts nodes have no public IP addresses, and the master node is reached through a bastion host.

For details, see Integration with cloud providers.

License key

The credential required for using the commercial editions of Deckhouse Platform.

The license key is required to pull images of the commercial editions from `registry.deckhouse.io`, for example when updating a cluster in a private environment with `d8 mirror pull`.

For details, see Updating DP in air-gapped environment.

Live migration

Moving a running virtual machine from one node to another without shutting it down.

Live migration is used for load balancing, to free a node for maintenance or update, and for a virtual machine firmware update that would otherwise require a restart. A node sends the memory of one virtual machine at a time and accepts one incoming migration at a time, and the transfer rate of a single migration is limited to 640 MiB/s. VM rebalancing also moves machines with live migration.

For details, see Maintenance of nodes running virtual machines.

Local storage

Storage on the local disks of a node. It reduces network latency and gives higher performance than storage accessed over the network.

Local storage is one of the software-defined storage types and is provided by the `sds-local-volume` module, which supports the LVM and LVMThin modes. It is effective in test environments and EDGE clusters.

For details, see Setting up local storage based on LVM.

Log buffering

Temporary accumulation of logs in a buffer before they are sent to the storage system.

Buffering keeps logs during temporary connection issues with the storage and sends them when the connection is restored, smooths load spikes on the storage and reduces the number of network requests by sending logs in batches. It is configured with the `buffer` parameter of ClusterLogDestination in the log pipeline.

For details, see Log collection and delivery.

Log pipeline

The chain of log collection, filtering and delivery that the `log-shipper` module builds from ClusterLoggingConfig, PodLoggingConfig and ClusterLogDestination.

ClusterLoggingConfig describes log sources at the cluster level, PodLoggingConfig describes sources within one namespace, and ClusterLogDestination defines the parameters of the log storage. Along the pipeline, logs can be filtered by labels and message content, enriched with Kubernetes metadata, buffered and sent to several storages at once, for example to the short-term log storage and to Elasticsearch.

For details, see Log collection and delivery.

Long-term metrics storage

A separate storage, for example VictoriaMetrics, to which Prometheus sends metrics with the remote_write mechanism.

In Deckhouse Platform, sending metrics to a long-term storage is configured with a PrometheusRemoteWrite object.

For details, see Writing Prometheus data to longterm storage.

Main cluster network

The network in which every virtual machine gets an IP address. It is bound to the main network interface of the pod that runs the machine.

In the list of networks of a virtual machine the main cluster network has the type `Main`. It does not have to be listed, but if it is, it must come first. Adding or removing the main network requires a reboot of the machine, unlike additional networks, because it cannot be changed on a running pod. The subnets that machines get addresses from are set in `virtualMachineCIDRs` of the `virtualization` module. When an address pool is configured for the main network in the `sdn` module, the addresses are managed by the shared IPAM.

For details, see IP addresses of virtual machines.

Main project namespace

The user namespace that is created together with the project. It is used to manage additional project namespaces and to run the workload.

The main project namespace is a user namespace created as part of a project. Additional project namespaces are managed from it.

Maintenance mode of a module or application

A mode in which Deckhouse Platform stops applying templates and restoring the resources of a module or an application, so that they can be changed manually. It is enabled with spec.maintenance: NoResourceReconciliation and is not related to the node maintenance mode.

In this mode changes of settings and values are not reflected in the resources, deleted resources are not restored, and hooks keep running. The resources rendered from the templates get the `maintenance.deckhouse.io/no-resource-reconciliation` label, and the admission policies stop protecting them. For an application instance the `Managed` condition becomes `False` and the `ApplicationIsInMaintenanceMode` alert fires. To return to normal operation, remove the `spec.maintenance` field.

The field is set in ModuleConfig for a module and in Application for an application. The mode is intended for debugging and development, not for production clusters. The node maintenance mode is a different mechanism.

For details, see Lifecycle and debugging.

Manage role

A role of the granular role model for administering Deckhouse Platform or one of its subsystems, that is, cluster-wide resources, the modules of the subsystem and their system namespaces.

Manage roles belong to the granular role model. A manage role grants access to cluster-wide Kubernetes resources, to the ModuleConfig and cluster-wide resources of the modules of its subsystem, and to the system namespaces in which these modules run. It does not grant access to the namespaces of user applications.

The name of a manage role has the form `d8:manage:<SUBSYSTEM>:<ACCESS_LEVEL>`, for example `d8:manage:networking:manager`. The subsystem `all` covers all modules. Permissions within a namespace are granted with use roles.

For details, see Granular authorization model.

Managed cluster

A Deckhouse Platform cluster with the `commander-agent` module installed. Its configuration and lifecycle are managed from a managing cluster.

The `commander-agent` module connects the cluster to the `commander` module of the managing cluster. Not to be confused with a Managed Kubernetes cluster, in which the control plane is provided by a cloud provider: see cluster managed by Deckhouse Platform.

For details, see Cluster management.

Managed Gateway

The Gateway object that the `alb` module creates from a ClusterALBInstance or an ALBInstance. Applications are published through it with ListenerSet and route resources of the Kubernetes Gateway API.

A cluster-wide gateway is created by the cluster administrator from a ClusterALBInstance, usually in the `d8-alb` namespace. A dedicated gateway in the application namespace is created from an ALBInstance. The name of the managed Gateway is published in the `status.gateway` field of the instance. The namespace administrator attaches ListenerSets with hostnames, TLS and ports, and developers attach HTTPRoute and other routes. The listeners `d8-http` and `d8-https` serve service tasks such as availability checks and are not meant for application routes.

For details, see Publishing applications using the Kubernetes Gateway API.

Managed service

A ready-to-use application service in the cluster, such as a database or a message queue, whose lifecycle is handled by Deckhouse Platform.

Each managed service adds its own Kubernetes resources and controllers to the cluster. A user describes the desired state of the service in a namespaced resource, and the controller creates and updates the related objects: workload instances, Services, secrets and snapshots. The administrator manages cluster-wide resources, such as service classes, that define the allowed configurations and default values for user resources. Deckhouse Platform handles administration, scaling, backup and updates of the service.

For details, see Managed services.

Managing cluster

A Deckhouse Platform cluster with the `commander` module installed. It creates standardised Deckhouse Platform clusters and manages their configuration and lifecycle.

The `commander` module implements a web application for cluster management and needs a PostgreSQL instance, deployed inside or outside the cluster. Clusters under its management run the `commander-agent` module and are called managed clusters.

For details, see Cluster management.

Marketplace

The package management system of Deckhouse Platform. Administrators connect package repositories, and users install applications from the packages in their namespaces.

A cluster administrator connects a package repository with PackageRepository, monitors the scanning

operations in which Marketplace discovers the available packages, and grants users the permissions to view package versions and to create Application objects. An installed application is an Application object in the namespace of the user.

Marketplace is available in Deckhouse Platform starting from version 1.76.

For details, see Marketplace.

Marketplace package

An abstract unit of Marketplace: either an application, a user workload deployed into a namespace, or a module that extends the capabilities of the cluster.

Packages are published in container registries connected to Marketplace as package repositories. For every discovered package version Deckhouse Platform stores its metadata: description, lifecycle stage, requirements, deletion confirmation, changelog and the OpenAPI schemas of settings and values.

For details, see Concepts.

Master node

A node that runs the control plane components. Master nodes carry the label `node-role.kubernetes.io/control-plane`, and the `control-plane-manager` module runs on them.

Deckhouse Platform manages the control plane on master nodes: certificates, component configuration, versions, the etcd cluster and kubeconfig files. In a cloud cluster, master nodes are CloudPermanent nodes.

The number of master nodes defines whether the cluster is single-master or multi-master. With more than one master node, HA mode is enabled automatically.

For details, see General management and configuration of the control plane.

Minor version

A Deckhouse Platform release that includes new features.

A minor version is one of the two types of a Deckhouse Platform release, the other being a patch version. Minor versions are installed sequentially: if the cluster lags behind the release channel by several minor versions, Deckhouse Platform reconstructs the intermediate releases, because they contain migrations that must be applied in order. In the `Manual` update mode a minor version needs approval.

For details, see Updating.

Module

A set of resources and applications that adds new capabilities to Deckhouse Platform.

Modules perform related functions, for example collecting metrics, balancing traffic or working with secrets. The documentation of a module and its API is available in the module library.

Module accessibility

The accessibility settings of module.yaml that define in which Deckhouse Platform editions a module is available and in which module bundles it is enabled by default.

Settings are given per edition with a `_default` block for editions without their own settings: `available` says whether the module can be used in the edition, `enabledInBundles` lists the module bundles (`Default`, `Managed`, `Minimal`) in which the module is enabled without a ModuleConfig.

For details, see Module structure.

Module artifacts

The three kinds of artifacts a module build pushes to the container registry, images of application containers, the module image and the release. Deckhouse Platform pulls and updates a module from them.

Images of application containers are tagged with content-based tags and referenced from the templates. The module image is built by `werf.yaml` and tagged with the semantic version. The release is the artifact of a module version: it carries a semantic version tag and a tag of the release channel, and Deckhouse Platform decides whether to update the module in the cluster by its data. The registry

with the artifacts serves as a module source.

For details, see Build and publish.

Module bundle

A named set of built-in Deckhouse Platform modules that are enabled by default. Default, Minimal or Managed.

The bundle is set in the `bundle` parameter of the `deckhouse` module and cannot be changed in a deployed cluster. `Default` is the recommended set for cluster operation, including monitoring, authorization, networking and other basic modules. `Managed` is the set for managed clusters of cloud providers. `Minimal` is the minimal set.

Regardless of the bundle, a module can be enabled or disabled explicitly in its ModuleConfig.

For details, see Deckhouse Platform configuration.

Module dependencies

A set of conditions (requirements) that must be met for Deckhouse Platform to run a module. They are declared in the requirements section of module.yaml.

A module can depend on the Deckhouse Platform version, on the Kubernetes version and on the versions of other modules. Dependencies on other modules affect the module startup order together with the module weight, and a dependency may be conditional. A Marketplace package declares the same kind of requirements in `package.yaml`, including groups of alternative and incompatible modules.

For details, see Dependencies of the Deckhouse Platform modules.

Module exclusive group

A group of modules that share one value of the exclusiveGroup parameter in module.yaml. Only one module of the group can be active in the cluster at a time.

Exclusive groups prevent conflicts between modules that perform similar or incompatible functions. Enabling a second module of the same group is not possible while the first one is active.

For details, see Module structure.

Module from a source

A module that Deckhouse Platform gets from a module source instead of its own distribution. Its release cycle is not tied to the release cycle of the platform.

The list of available modules is defined by the module sources tracked by Deckhouse Platform. Such a module is updated by its own update policy: it has its own release channels, lifecycle stage and dependencies. Alerts of modules from a source are not included in the alerts reference of the platform.

For details, see Updating DP in air-gapped environment.

Module hook

An executable file of a module or a Marketplace package that Deckhouse Platform runs in response to an event. Hooks prepare values for templates, create and change objects in the cluster and validate settings.

A hook runs when one of its bindings fires: at startup, before or after the Helm templates are applied, on schedule or when watched Kubernetes objects change. The hooks of a module are located in the `/hooks/batch` directory. The hooks of a Marketplace package are written in Go with module-sdk and compiled into a single binary. Module hooks and global hooks exist. The Deckhouse controller runs them as tasks of the addon-operator queues.

For details, see Module structure.

Module lifecycle stage

The maturity stage of a module or a Marketplace package, Experimental, Preview, General Availability or Deprecated. It is declared in module.yaml or package.yaml and, together with the release channel, defines how stable a version is.

- `Experimental`: the functionality may change significantly and compatibility with future versions is

not guaranteed; such modules cannot be enabled unless the `allowExperimentalModules` parameter of the `deckhouse` module is set to `true`.
- `Preview`: a preliminary version; the basic features are preserved, compatibility is ensured but may require migration.
- `General Availability`: ready for production environments.
- `Deprecated`: development and support are discontinued.

The stage of a Marketplace package is published in the package metadata when Deckhouse Platform scans the repository.

For details, see Versioning modules.

Module settings conversion

A rule in the `openapi/conversions` directory of a module that converts module parameters from one version of the OpenAPI specification to the next, for example when a parameter is renamed or moved.

Each conversion works between two consecutive versions and is stored in a `v<N>.yaml` file with its tests. The chain of conversions must cover all versions of the parameter specification without gaps, so that Deckhouse Platform can convert the settings of a module automatically when the version of its settings schema changes.

For details, see Module structure.

Module source

The container registry from which Deckhouse Platform fetches information about modules that are not built into the platform. It is set by ModuleSource with the registry address and authentication parameters.

Modules from a source are published to the registry as module artifacts. Deckhouse Platform synchronizes with the source and creates module releases according to the module update policy. A module is then enabled with ModuleConfig. In an air-gapped cluster the source points to the in-cluster or private registry.

For details, see How to run and verify a module in the DP cluster.

Module update policy

The rules by which Deckhouse Platform updates a module from a source in the cluster. It is set by ModuleUpdatePolicy and defines the update mode, the release channel and the update windows of the module.

A policy sets the update mode (automatic, manual or updates disabled), the release channel used for updates and the update windows in which automatic updates are allowed. If no policy is defined for a module, the module follows the update settings of the platform from the `update` parameter of the `deckhouse` module. A module release in the `Pending` status waits for manual approval.

For details, see How to run and verify a module in the DP cluster.

Module weight

A number from the weight parameter of module.yaml that defines the module startup order. Modules with a lower weight start earlier, the default is 900.

If `module.yaml` is missing, the weight is taken from the numeric prefix of the module directory, for example `040-node-manager` means 40. If it cannot be obtained, 100 is used. The Deckhouse controller considers the weight only for critical modules, whose tasks go to the `main` queue in ascending order of weight. The startup order is also affected by module dependencies.

For details, see Queuing mechanism.

Monitoring node

A dedicated node for Grafana, Prometheus and other monitoring components. Without such nodes the monitoring components run on system nodes.

Dedicated monitoring nodes are recommended in heavily loaded clusters with many alerts and large volumes of metrics. They need fast disks, at least 400 IOPS. Monitoring nodes usually form the `monitoring` node group.

For details, see Recommendations for configuring cluster nodes.

Multi-master cluster

A control plane operation mode with several master nodes. kube-apiserver interacts with all etcd instances, and requests are forwarded to other nodes if the local kube-apiserver is unavailable.

The multi-master mode provides high availability of the control plane and supports scaling of master nodes. When the cluster has more than one master node, HA mode is enabled automatically. In a configuration with two master nodes, an arbiter node provides the etcd quorum.

For details, see Scaling and changing master nodes.

Multicluster

An inter-cluster communication model that tightly connects clusters. The clusters share namespaces, and the services of one cluster are available to the others as local ones.

Access to services from other clusters can be restricted by authorization policies. The model is implemented with Istio through the `istio` module and is built on the service mesh. The model for independent clusters is federation.

For details, see Inter-cluster cooperation.

Namespace

A Kubernetes mechanism for grouping objects logically.

Examples of Kubernetes objects that can be grouped by a namespace: Pod, Deployment, Service, Ingress, ConfigMap.

Namespaces are often used for isolation, but without additional configuration they do not provide it, unlike projects in Deckhouse Platform.

Namespaces are divided into system and user namespaces.

Network policy

Rules that restrict how pods communicate with each other, with external systems and with cluster nodes.

The implementation of network policies in Deckhouse Platform depends on the enabled CNI module. With `cni-cilium`, policies are enforced by Cilium in three formats: the standard NetworkPolicy at L3 and L4, Cilium network policies with L3–L7 rules within a namespace, and CiliumClusterwideNetworkPolicy for the whole cluster, which also protects nodes as a host firewall. With `cni-flannel` or another CNI without policy support, the `network-policy-engine` module enforces the standard NetworkPolicy at L3 and L4 through `iptables` and `ipset` rules on every node.

For details, see Network policies.

NLB (network load balancing)

Load balancing at the network level. Traffic is routed by IP addresses and ports without inspecting the contents of requests.

In Deckhouse Platform, NLB is implemented either by an external load balancer of the cloud provider or by the built-in `metallb` module, which works both in cloud and in bare-metal clusters. Balancing at the application level is ALB.

For details, see Incoming traffic balancing.

Node group

A set of cluster nodes with common parameters, described by a NodeGroup. The NodeGroup defines the node type, the number of nodes and their configuration, and Deckhouse Platform keeps the nodes in line with it.

Node groups divide the cluster infrastructure logically. Typical groups are `master` for the control plane, `front` for frontend nodes, `monitoring` for monitoring nodes, `system` for system nodes and `worker` for user applications.

Each group has centralized settings: the Kubernetes version, resources, taints, labels and kubelet parameters. When a NodeGroup is created or changed, the `node-manager` module brings the nodes of the group to the specified configuration. The node type is one of Static, CloudEphemeral, CloudPermanent or CloudStatic.

For details, see Node management basics.

Node group autoscaling

Automatic change of the number of nodes in a group by the Cluster Autoscaler component of the `node-manager` module, based on CPU and memory demand. It triggers only when Pending pods cannot be scheduled on the existing nodes.

Autoscaling is available for CloudEphemeral nodes. The limits are set in the `cloudInstances` section of the node group: `minPerZone` and `maxPerZone` bound the number of nodes per zone, `maxUnavailablePerZone` limits unavailable nodes during changes, `priority` defines which groups Cluster Autoscaler scales first. Autoscaling does not apply to static nodes: for them CAPS maintains the number set by `staticInstances.count`.

For details, see Adding and managing cloud nodes.

Node maintenance mode

A node state in which Deckhouse Platform moves the virtual machines to other nodes and does not start new ones on it, so that work on the node does not disrupt them.

A node is switched to maintenance mode with the `d8 k drain` command or with **Cordon + Drain** in the web interface, and returned to service with `d8 k uncordon`. Virtual machines that can be moved are moved by live migration. A machine that cannot be moved, for example one pinned to the node or using a passed-through device, keeps the node busy until it is restarted. The restart is allowed by the `virtualization.deckhouse.io/virtualmachines-restart-approved` annotation on the node.

For details, see Maintenance of nodes running virtual machines.

Orphan resources of an application

Objects that stay in the namespace after the Helm release of an application is uninstalled, for example objects created by workloads or hooks at runtime. Deckhouse Platform deletes the listed kinds together with the Application object.

The package declares the kinds of orphan resources in the `orphanResources` field of `openapi/values.yaml`. When the Application is deleted, Deckhouse Platform deletes the objects of these kinds in the namespace of the application instance that carry the `packages.deckhouse.io/instance: <INSTANCE_NAME>` label, after the Helm release is uninstalled and the hooks are stopped. Objects created by workloads and hooks do not get the label automatically.

Orphan resources are deleted only when the Application is deleted. On suspension and on a package version change they stay in the cluster.

For details, see Lifecycle and debugging.

Package bundle

The main image of a package version in the container registry. It contains the package manifest, image digests, OpenAPI schemas, Helm templates, the hooks binary, documentation and the changelog.

The bundle is published as `<REPOSITORY>/<PACKAGE_NAME>:<PACKAGE_VERSION>`, next to the container images of the application (referenced by digest), the version metadata image and the release channel images. Deckhouse Platform downloads the bundle of the requested version as the first step of installing an application instance, then reads `package.yaml`, the schemas and `images_digests.json` and discovers the hooks.

For details, see Application development.

Patch version

A Deckhouse Platform release that includes fixes. Patch versions are published when needed.

A patch version is one of the two types of a Deckhouse Platform release, the other being a minor version. In the `AutoPatch` update mode only patch versions are installed automatically. Starting from Deckhouse Platform 1.70 they are applied with update windows taken into account.

For details, see Updating.

Pod limit per node

The maximum number of pods that can be placed on a node. In Deckhouse Platform it is derived from the node subnet size set by the podSubnetNodeCIDRPrefix parameter when the cluster is deployed.

With a prefix of 24 or larger a node holds 120 pods, with 23 it holds 250, with 22 it holds 500, and with 21 or smaller it holds 1000. The parameter is part of the protected parameters of the cluster, so choose the value at deployment time if many pods per node are expected. The control plane components for high pod density are configured by Deckhouse Platform automatically.

For details, see High pod density.

Pod redistribution

Periodic eviction of pods that match the active scheduling strategies, so that they are scheduled again according to the current state of the cluster. Controlled by the descheduler module.

Deckhouse Platform analyzes the state of the cluster every 15 minutes and evicts the matching pods. The evicted pods go through the standard scheduling process again. The priority class of a pod defines the order of eviction, and the `priorityClassThreshold` parameter of a Descheduler limits eviction to pods with a priority below the threshold. VM rebalancing is built on the same mechanism.

For details, see Scheduler.

Pod Security Standards

Three levels of pod security policies, privileged, baseline and restricted, that Deckhouse Platform applies to namespaces with the `admission-policy-engine` module.

Pod Security Standards are one of the categories of admission policies. The level `privileged` imposes no restrictions, `baseline` prevents the most common privilege escalation techniques, `restricted` sets the strictest requirements for pods. Deckhouse Platform takes only the policy descriptions from Kubernetes and enforces them with Gatekeeper rather than with the Kubernetes Pod Security Admission controller.

The default level is `baseline`. It applies to namespaces without the `security.deckhouse.io/pod-policy` label, except the system ones. The label assigns a level to one namespace, the `admission-policy-engine` module setting changes the default.

For details, see Security policies.

Post-bootstrap script

A custom script that the installer runs on one of the master nodes after the cluster installation is complete.

The script is used for additional cluster configuration, collecting diagnostic information or integration with external systems. Its path is passed to the installer with the `--post-bootstrap-script-path` parameter.

For details, see Installation.

Preflight checks

Checks that the installer runs before installing Deckhouse Platform.

The checks are grouped into general checks, checks for static clusters and checks for cloud clusters. An individual check is skipped with the `--preflight-skip-check` flag and its name, `--preflight-skip-all-checks` skips all of them, and `dhctl preflight list` prints the full list. Cluster subnets and the DNS name template are validated when the installation configuration file is loaded and cannot be skipped.

For details, see Installation.

Priority class

One of the PriorityClass objects that Deckhouse Platform creates in the cluster and assigns to its components. When resources are insufficient, the scheduler preempts pods with a lower priority first.

Applications use a priority class through the `priorityClassName` field. A pod without it is treated as having the lowest priority. The classes range from `system-node-critical` and `system-cluster-critical`, reserved for critical cluster components, through `production-high`, `production-medium`, `production-low` and `staging` to `develop`, the default. Additional classes can be created with a name and a numeric priority value. Priority classes also define the order of eviction during pod redistribution.

For details, see Priority classes.

Private environment

An environment without direct access to the public container registry of Deckhouse Platform and to external package repositories. Access goes through a proxy server or through a container registry deployed inside the perimeter.

Proxy server parameters set in the cluster configuration during installation are propagated to the environment variables of cluster nodes and Deckhouse Platform components. User applications do not receive them automatically. Cluster nodes are usually reached through a bastion host, and the private registry runs on a separate machine in the internal network. When access to external resources is disabled completely, no proxy is used, the required artefacts are brought into the perimeter by other means, and the cluster is an air-gapped cluster.

For details, see Installing DP in a private environment.

Project

A mechanism for isolating and managing workloads in Deckhouse Platform. A project logically divides the cluster and gives a team its own area to deploy applications, configure resources and manage access within the granted rights.

Projects provide isolation, support creation from templates, control which platform capabilities are available to the team and are compatible with Kubernetes namespaces. A project includes the main project namespace and may include additional project namespaces.

For details, see Multitenancy.

Project features

The set of platform capabilities available within a project. Features define which APIs and resource kinds can be used in the project.

Project features are not a part of the role model. Examples of features: Kubernetes, Virtualization. The features of a single namespace within the project are limited by the project namespace features.

Project namespace features

A restriction of the project features at the level of a single user namespace within the project.

The set of features of a user namespace cannot exceed the set of features of its project.

Project network

An additional software-defined network available within one namespace. The user creates and manages it with the NetworkClass provided by the administrator. It is also called a user network.

The administrator defines in a NetworkClass the range of VLAN tags and the node network interfaces on which users may create their own networks. The user creates the network with the Network resource in the namespace of the project. Users can allocate an address pool for the project network and assign addresses to pod interfaces through IPAM. The other kind of additional network is the cluster network.

For details, see Additional networks for use in application pods.

Protected parameters

Cluster parameters that are critical for cluster operation and cannot be changed in a running cluster by default. podSubnetCIDR, podSubnetNodeCIDRPrefix and serviceSubnetCIDR.

The protected parameters are part of the cluster configuration. An attempt to change them is blocked by the admission webhook with an error message. The procedure for changing them, with the installer or manually, is described in the documentation.

For details, see Deckhouse Platform configuration.

Quorum

The majority of the members of a distributed system (more than half of their total number) that must be available for the system to keep working and accept changes.

In Deckhouse Platform, the quorum matters most for etcd, the storage of the control plane. The etcd cluster accepts changes only while the majority of its members (N/2 + 1) are available: a cluster of three members tolerates the failure of one member, a cluster of five tolerates the failure of two. An even number of members does not improve fault tolerance: with two members, the failure of either one loses the quorum. For this reason, use an odd number of master nodes, usually three. In a configuration with two master nodes, the third etcd member may run on an arbiter node, which provides the quorum.

For details, see High availability in the control plane recovery guide.

Region

An independent geographic area that includes several, typically at least three, isolated availability zones united into a single logical infrastructure unit with shared management services.

A region is a key element of the global distributed architecture of cloud providers. Regions are widely dispersed geographically, placed outside seismically or politically unstable areas and connected to several power grids. Latency between the availability zones of one region is about 5 ms, between regions it is 50 to 200 ms.

Deckhouse Platform supports distributing cluster nodes across regions (Multi-Region) so that applications keep running even when a whole data center or cloud provider fails. Because of the latency between regions, this is appropriate only for applications that work independently in each region, for example CDN cache servers or distributed monitoring nodes. The administrator labels each node with its region and provides stable communication channels between regions.

For details, see Geo-distribution.

Registry management mode

The mode in which the registry module manages the pull paths of Deckhouse Platform component images, Unmanaged or Managed.

The mode is set by the `mode` parameter in the `registry` ModuleConfig:

- `Unmanaged` (the default). The `registry` module does not manage the pull paths: the cluster pulls images from the registry it was installed with.
- `Managed`. The `registry` module manages the pull paths. The `primary.upstream` parameter sets the upstream registry, and `storage.cache` enables the in-cluster image cache on the master nodes. Without `primary.upstream`, the cluster is considered isolated.

For details, see Managing the registry in DP-managed clusters.

Release channel

One of the five channels, from Alpha to Rock Solid, through which every new Deckhouse Platform version is rolled out gradually. The cluster receives the versions published on the channel it follows.

Each new version is first published to the Alpha channel and then moves to Beta, Early Access, Stable and Rock Solid. Versions appear on Early Access at least 7 days after the release, on Stable at least 14 days and on Rock Solid at least 30 days after the release.

The channel of the cluster is set in the `settings.releaseChannel` parameter of the `deckhouse` module. How a new version is applied depends on the update mode and update windows.

For details, see Updating.

Release channel image

The release-channel image in the Deckhouse Platform registry whose tag is the name of a release channel. The tag points to the image of a specific Deckhouse Platform version and is moved to a new image when a new version is released.

The image contains `version.json` with the current version of the release channel and `changelog.yaml`. When the version in the image changes, Deckhouse Platform creates a DeckhouseRelease object and starts the update. A rollout can be paused by patching the image with a `suspend` flag, which makes it a suspended release. In a private environment the image is copied to the private registry together with the image bundle.

For details, see Updating DP in air-gapped environment.

Replicated storage

Storage in which data is replicated across several nodes, so that it stays available when one of the nodes fails.

Replicated block StorageClass objects are created with the `sds-replicated-volume` module, which is based on Distributed Replicated Block Device (DRBD). Replicated storage is one of the software-defined storage types. Its pools are created on LVM volume groups as thick or thin pools.

For details, see Setting up replicated storage based on DRBD.

Role model

Role-based access control (RBAC), a mechanism that defines which users or services can perform specific actions within projects or the system.

Deckhouse Platform supports two role models: the granular one, which is recommended and is built on use roles and manage roles, and the basic one with high-level roles. Both models are supported by the `user-authz` module and can be used at the same time: the permissions they grant are summed up.

For details, see Authorization.

Runtime audit

Detection of security threats in a running cluster by analysing Linux kernel events and Kubernetes API audit events.

Runtime audit in Deckhouse Platform uses two sources of events: Linux kernel events, received through the eBPF driver of the Falco threat detection system, and Kubernetes API audit events, received through a webhook. It detects attempts to exploit known vulnerabilities, signs of cryptocurrency miners and Kubernetes-specific threats such as shells launched in containers, privileged containers, mounting of insecure paths and reading of sensitive files.

Capturing kernel events requires Linux kernel 5.8 or later with eBPF support.

For details, see Runtime audit.

Runtime audit rule

A rule of the runtime audit that describes a pattern of suspicious behaviour as a condition expression in the Falco syntax. Deckhouse Platform ships built-in rules and accepts custom ones.

Built-in rules include rules for the Kubernetes audit, which reveal problems with the security of Deckhouse Platform and of the audit mechanism itself, and the `fstec` regulatory rules that meet the requirements of FSTEC of Russia order No. 118 for containerization tools. Custom rules are defined with a FalcoAuditRules object of the `runtime-audit-engine` module. The runtime audit matches events against all rules and produces security events.

For details, see Runtime audit architecture.

Runtime profile

A behaviour model of an application or container at runtime, including system calls, executed processes, file access and network interactions, built by the node-agent of the `operator-trivy` module with eBPF programs.

The node-agent is an optional component that runs on all cluster nodes in privileged mode, observes container behaviour with eBPF programs and stores the profiles in the ApplicationProfile and NetworkNeighborhood objects. It works in profiling mode and does not detect attacks by rules. That is the task of the runtime audit. The profiles can be used to generate network policies from the observed traffic.

For details, see Module operator-trivy.

Scheduler extender

An extension of kube-scheduler provided by a Deckhouse Platform module that places pods according to the module’s own logic instead of the standard scheduler, for example by local volumes, node selectors of a storage class or network interfaces.

The `sds-local-volume` module places pods with local volumes by the free space on nodes and the space to reserve for the volumes. Its webhook sets `spec.schedulerName` of such pods to `sds-local-volume`. The `csi-nfs` module places pods by the node selectors of NFSStorageClass, and the `sdn` module suggests nodes according to the network interface settings of pods using software-defined networks. kube-scheduler sends scheduling requests for such pods to the extender’s webhook.

For details, see Storage architecture.

Secret distribution

Automatic copying of secrets from the default namespace that carry the `secret-copier.deckhouse.io/enabled` label to all namespaces of the cluster.

Deckhouse Platform copies the labelled secrets to every existing and every new namespace,

propagates updates of the original, overwrites manual changes of the copies and deletes the copies when the original is deleted. A nightly synchronization keeps the copies up to date. This removes manual copying of secrets in CI/CD pipelines, for example registry credentials.

The mechanism is not compatible with the multitenancy mode: a secret distributed to all namespaces can reach the isolated environment of a project. Shared secrets for projects are defined in the project template instead.

For details, see Automatic distribution of secrets across namespaces.

Secrets store

An external store compatible with the HashiCorp Vault API, or the built-in Deckhouse Stronghold, from which the `secrets-store-integration` module delivers secrets to applications without storing them in etcd.

The `secrets-store-integration` module delivers secrets to pods as mounted files or environment variables, refreshes mounted secrets every two minutes when the value in the store changes and injects an entrypoint for applications that cannot read secrets from the store themselves. It connects to an internal Deckhouse Stronghold instance automatically in the `DiscoverLocalStronghold` mode and works with any Vault-compatible store in the `Manual` mode.

For details, see Secrets-store-integration module.

Security event

A structured record of an action or fact that is significant from an information security perspective, extracted from the logs of applications and Kubernetes infrastructure components.

Deckhouse Platform collects security events from container sources, that is, application logs in pods, and from cluster sources, such as the Kubernetes API audit log and node authentication logs. The `log-shipper` module pre-selects log records, and the `security-events-manager` module parses them, brings them to a unified format with a mandatory set of attributes, enriches them with context and delivers them to storage and analytics systems such as Loki, Elasticsearch, Kafka or Splunk.

For details, see Security events.

Service mesh

An infrastructure layer that manages network communication between the services of an application. Encryption, access control, routing, request retries and metrics collection are configured centrally, without changing the code of the services.

The services of an application exchange requests over the network, and without a service mesh each of them has to implement encryption, authorization, retries and timeouts on its own. A service mesh takes over these tasks: traffic between services passes through proxies that are configured from a single place. Usually the proxy is added to the application pod as a sidecar container. In the experimental ambient mode the proxy runs at the node level, and pods need no sidecars.

In Deckhouse Platform the service mesh is provided by the `istio` module. Within a single cluster, it is used, for example, for canary deployment and for automatically cutting off a failing service (Circuit Breaker). It also forms the basis for federation and multicluster.

For details, see Inter-cluster cooperation.

Short-term log storage

The built-in in-cluster log storage of Deckhouse Platform based on Grafana Loki. It is integrated with log collection and added to Grafana as a data source.

The storage is provided by the `loki` module, whose settings define the disk size, the retention period, the StorageClass and the resources. Logs of system components flow into it after the log pipeline is configured. Log collection from user applications is configured separately. The storage does not support the high availability mode, so important logs are kept long-term in an external storage.

For details, see Short-term log storage.

Single-master cluster

A control plane operation mode with one master node. kube-apiserver uses only the local etcd instance and listens only on the address of the master node.

A proxy server on the node handles requests on `localhost`. When master nodes are added or

removed, Deckhouse Platform migrates the control plane between the single-master and multi-master modes.

For details, see Scaling and changing master nodes.

Single-node cluster

A cluster of one node that performs all node roles. It has no fault tolerance and is intended for familiarisation or workloads with low resource requirements.

After the taint is removed from the node, it runs all cluster components of the selected module bundle. The resource requirements for such a node depend on the workload type: containers, virtual machines or both. Do not use single-node clusters in production environments.

For details, see the Resource requirements section of the “System requirements” page.

Sizing policy

The block of a virtual machine class that defines which combinations of cores, core fraction and memory are allowed for virtual machines of this class.

A policy consists of a list of rules, each applying to its own range of cores. The ranges must not overlap. Changes to the policy affect existing virtual machines: a machine that no longer meets the requirements gets the `SizingPolicyMatched` condition with the `False` status. The allowed values of the core fraction set in the policy define the degree of CPU oversubscription.

For details, see Virtual machine sizing policy.

Skipping intermediate module releases

The from-to mechanism that lets Deckhouse Platform skip step-by-step module updates. If the installed module version falls into a from-to range declared in module.yaml, Deckhouse Platform installs the latest version within to.

Transition rules are listed in the `update.versions` section of `module.yaml`. A release that contains them is called a constrained release. The rule is read from the release to move to, and its `to` must match the version of that release. If the installed version is lower than `from`, the update proceeds sequentially. When several rules match, the one with the largest `to` is chosen.

For details, see Module development and debugging.

SLA monitoring

Collection of availability statistics of cluster components and Deckhouse Platform components by the upmeter module to evaluate SLA compliance.

Availability data is shown in the web interface and can be exported over the Prometheus Remote Write protocol with an UpmeterRemoteWrite object.

For details, see Cluster SLA monitoring.

Software-defined network (SDN)

An additional network segment for pods and virtual machines that the administrator describes declaratively in custom resources, while the `sdn` module configures the nodes.

Software-defined networks in Deckhouse Platform cover the configuration of network interfaces on nodes (port aggregation, bridges, VLAN interfaces), additional networks for workloads that are available in the whole cluster or within one namespace (project networks), IP address management for these networks, underlay networks for passing physical interfaces to applications, and system networks for service traffic at the node level.

For details, see Software-defined networking.

Software-defined storage

A group of storage types in Deckhouse Platform that run on the disks of the cluster nodes and are managed by the sds-* modules, as opposed to external storage systems.

Software-defined storage (SDS) includes managing logical volumes on cluster nodes, local storage based on Local Path Provisioner or LVM, and replicated storage based on DRBD. The other group is external storage: Ceph, NFS, S3 and storage systems of hardware vendors.

For details, see Storage.

Standby node

A pre-provisioned node kept in the cluster so that new application replicas start without waiting for node creation during horizontal scaling. The number of standby nodes is set by cloudInstances.standby.

Standby nodes are available for CloudEphemeral nodes. The number is set as an absolute value or as a percentage of the maximum number of nodes in the node group. At least one node of the group must already be running. The share of node resources taken by service components outside Deckhouse Platform is set by `standbyHolder.overprovisioningRate`. Standby nodes increase infrastructure costs.

For details, see Adding and managing cloud nodes.

Static cluster

A cluster with the clusterType: Static. Its nodes are physical servers or virtual machines that Deckhouse Platform does not create or manage through a cloud API.

The administrator prepares the servers, and the installer configures them, deploys Kubernetes and installs Deckhouse Platform. The nodes of such a cluster are static nodes. A static cluster can be extended with nodes of a cloud provider into a hybrid cluster.

For details, see Platform installation.

Static node

A node on a bare-metal server or a virtual machine that is created outside Deckhouse Platform automation and connected to the cluster manually or through CAPS. It is described by a NodeGroup with the Static node type.

Static nodes do not support automatic scaling and updates: kubelet updates, restarts and replacements are performed manually or by external tools. A node is connected either by running the bootstrap script on the server or automatically by CAPS. In a cloud environment, cloud-controller-manager does not manage a Static node even if a cloud provider module is enabled.

A node group can combine static nodes with CloudStatic nodes, which is the basis of a hybrid cluster.

For details, see Adding and managing bare-metal nodes.

Static pod

A pod that kubelet starts directly from a manifest in the `/etc/kubernetes/manifests` directory of the node, without the Kubernetes API. Control plane components and kubernetes-api-proxy run this way in Deckhouse Platform.

In Deckhouse Platform kubelet processes only files with the `.yaml` or `.yml` extension in this directory. Backup copies, editor swap files and other temporary files are ignored. The control plane components kube-apiserver, etcd, kube-controller-manager and kube-scheduler and the kubernetes-api-proxy are static pods.

For details, see Kubelet.

Static user

A user created in the cluster as a User object of local authentication, with an email and a hashed password, without an external authentication provider.

Static users and groups are the basis of local authentication in Deckhouse Platform. A User object stores the user information and the password hash, the password is not stored in plain text. A Group object defines a list of users. Local authentication supports password policies and temporary users with a TTL. Two-factor authentication is also available.

The recommended way to create a static user is the `d8 iam user create` command.

For details, see Local authentication.

Stronghold lease

Temporary access to secrets and resources in the Deckhouse Stronghold secrets store, identified by a lease ID.

Leases are managed in the “Leases” section of the Stronghold web interface, where a lease is found by its ID. Together with access control policies, authentication methods and entities, leases define who may use the secrets store and for how long.

For details, see Secret management web interface.

Stronghold sealing

The state of the Deckhouse Stronghold secrets store in which it can neither read nor write secrets. The store is sealed and unsealed in the Stronghold web interface.

The “Seal Stronghold” section of the web interface shows the current state of the secrets store and a button that seals or unseals it. While the store is sealed, the `stronghold` module does not serve requests to read or write secrets.

For details, see Secret management web interface.

Stronghold secrets engine

A component of the Deckhouse Stronghold secrets store that holds and serves secrets of one kind, for example key-value secrets (KV) or certificates (PKI).

Secrets engines are managed in the Stronghold web interface: each engine shows its list of secrets (roles, keys and others), its configuration and a button to add a new secret. The web interface is provided by the `stronghold` module and opens at `stronghold.<CLUSTER_NAME_TEMPLATE>`.

For details, see Secret management web interface.

Subsystem

A logical part of the system that groups related modules and their system components.

Subsystems group the components of the system and delimit access to them. A subsystem consists of modules and their system components.

Suspended release

A Deckhouse Platform release whose rollout on a release channel was paused after it had been published. The release channel image is patched with a suspend flag.

While a release is suspended, `d8 mirror pull` refuses to copy the release channel and reports that the source registry contains a suspended release channel. The platform images can still be downloaded by specifying a version explicitly with the `--deckhouse-tag` flag.

For details, see Updating DP in air-gapped environment.

System

The part of the cluster that keeps it operational, manages resources and automates processes.

The system consists of subsystems, which in turn consist of modules and system components.

The system runs mostly in system namespaces. The parts of the system that serve the workload of a specific project can run in user namespaces.

System component

A process or a group of processes that run in a container or as a systemd service on a node and keep the platform and its modules running.

Examples of system components are kubelet and containerd.

Some system components run in user namespaces. For example, when a user creates a virtual machine or a managed service, the pods that support this object start in the user namespace.

System namespace

A namespace in which the system runs, that is, modules and system components. Running the workload in system namespaces is prohibited.

System namespaces of Deckhouse Platform have the prefixes `d8-` and `kube-`. The system, its modules and system components run there.

System network

An additional service network on the cluster nodes, created by the `sdn` module on top of an existing underlay network for node-level service traffic.

A system network is created with a SystemNetwork object. Interfaces connected to a system network get cluster-scoped addresses through ClusterIPAddress, like interfaces of cluster networks. It belongs to the node-level functions of software-defined networks, together with the configuration of node interfaces and underlay networks.

For details, see Software-defined networking (SDN).

System node

A node dedicated to running Deckhouse Platform modules so that they do not interfere with user applications.

System nodes are isolated with labels and taints so that user pods are not scheduled on them. Two system nodes are recommended. System nodes can be static or cloud nodes and usually form the `system` node group. Without dedicated monitoring nodes, the monitoring components also run on system nodes.

For details, see Recommendations for configuring cluster nodes.

The deckhouse system user

The local user and group with UID and GID 64535 that Deckhouse Platform components use to run unprivileged processes on nodes. The name and identifiers are reserved and not configurable.

If the account does not exist, bashible creates it without a password, `sudo` privileges or a login shell. The account must not be provided through AD or LDAP. Processes owned by `deckhouse` in `ps` or `top` output belong to Deckhouse Platform components. The preflight check accepts an absent user and group or an existing pair with the reserved identifiers and no `sudo` privileges. The installation fails if the name is used with other identifiers or the identifiers belong to another account.

For details, see Installation.

Thick volume

An LVM volume in a thick pool, that is, a pool of the LVM type: the space for such a volume is allocated in full at creation. A thick pool gives performance comparable to the storage device but does not allow snapshots.

Thick volumes are created in thick pools of local and replicated storage: the LVM mode of the `sds-local-volume` module and LVM volume groups without a thin pool definition in the `sds-replicated-volume` module. For volumes that need snapshots, use a thin volume.

Thin volume

An LVM volume in a thin pool, that is, a pool of the LVMThin type: the space for such a volume is allocated as it fills up. A thin pool allows snapshots but has lower performance.

Thin volumes are created in thin pools of local and replicated storage: the LVMThin mode of the `sds-local-volume` module and LVM volume groups with a thin pool definition in the `sds-replicated-volume` module. Where snapshots are not needed, a thick volume gives higher performance.

Underlay network

Direct connection of physical functions (PF) and virtual functions (VF) of the network interfaces of a node to pods through Kubernetes Dynamic Resource Allocation. It serves workloads that need direct access to hardware, such as DPDK applications.

Physical network interfaces are connected to pods in one of two modes: `Shared`, where Virtual Functions are created from a Physical Function with SR-IOV and several pods share the hardware, or `Dedicated`, where a pod gets the whole Physical Function. The administrator creates an UnderlayNetwork object of the `sdn` module and marks the namespace that may use it. The pod requests a device with an annotation.

DPDK, Data Plane Development Kit, is a set of libraries for processing network packets in user space, bypassing the kernel network stack.

For details, see Connecting physical network interfaces to DPDK application pods.

Update mode

The setting that defines how Deckhouse Platform applies new versions.

In the `Auto` mode both patch and minor versions are applied automatically, in the `AutoPatch` mode only patch versions within the current minor version. In the `Manual` mode a version is applied only after the administrator approves the corresponding DeckhouseRelease.

The mode is set in the `settings.update.mode` parameter of the `deckhouse` module. In the automatic modes updates are applied with regard to update windows.

For details, see Configuring updates.

Update window

A time interval during which Deckhouse Platform is allowed to install updates automatically. Outside the windows new versions are not installed.

Update windows are set in the `update.windows` parameter of the `deckhouse` module. If no windows are configured, an update is applied as soon as a new version appears on the release channel.

Disruptive updates of nodes have separate windows in the NodeGroup parameters `disruptions.automatic.windows` and `disruptions.rollingUpdate.windows`.

For details, see Configuring updates.

Use role

A role of the granular role model for granting users permissions within one namespace, that is, access to namespaced module resources and standard namespaced Kubernetes resources.

Use roles belong to the granular role model. They are intended for users, typically application developers, who do not manage the cluster or its modules but need to create Ingress resources, configure authentication for applications or collect logs in their namespace.

The `user-authz` module creates the roles `d8:use:role:viewer`, `d8:use:role:user`, `d8:use:role:manager` and `d8:use:role:admin`. A use role is granted only with RoleBinding. Administrator permissions are granted with manage roles.

For details, see Granular authorization model.

User namespace

A namespace in which the workload runs. It always belongs to one project.

A user namespace always belongs to one project and cannot have the prefix of a system namespace.

The namespaces of a project are divided into the main project namespace and additional project namespaces.

Virtual machine (VM)

A machine that Deckhouse Platform creates and manages declaratively through the virtualization module. It runs in a pod on a cluster node under QEMU/KVM, managed by KubeVirt.

A machine is described by VirtualMachine in a project or namespace. Its disks and images are attached as block devices, its CPU is set by the number of cores and the core fraction within the limits of the VM class, and its state is kept by the run policy. The guest system is configured on the first boot by provisioning, after which Deckhouse Platform communicates with it through the guest OS agent.

The module core is based on KubeVirt: the QEMU process started by libvirtd in the `compute` container of the machine pod is the running machine. Machines move between nodes by live migration, are grouped into pools, and go through the phases of the virtual machine lifecycle.

For details, see Virtualization subsystem.

Virtual machine block device

A disk or an image attached to a virtual machine. Images are attached read-only or as a CD-ROM for ISO images, disks are attached read-write.

Devices are attached in two ways. The `blockDeviceRefs` list of the virtual machine specification sets the boot order by position or by the `bootOrder` field and is used for manual configuration, for example an ISO for OS installation. A VirtualMachineBlockDeviceAttachment attaches a disk as a separate

resource that does not take part in the boot order, over the `virtio-scsi` bus, and is used for automation and when the user has no rights to edit the machine.

With paravirtualization enabled, both ways attach and detach disks on a running machine without a reboot, if the storage is available on the node where the machine runs. While a live migration prepares the target node, devices can be neither attached nor detached.

For details, see Attaching disks and images to a virtual machine.

Virtual machine clone

A copy of a virtual machine created from an existing machine or from its snapshot. The clone gets a new IP address and new MAC addresses, and labels are not copied to it.

Because a clone copies the machine, the resources of the new machine need unique names: `nameReplacements` replaces the names of specific resources, and `customization` adds a prefix or a suffix to the names of all cloned resources, such as disks and IP addresses.

The clone gets a new IP address in the cluster network and new MAC addresses for additional interfaces, so the network settings of the guest OS have to be reconfigured. Labels are not copied, so the clone does not receive Service traffic until the needed labels are added. A snapshot of the machine can be the source of the clone.

For details, see Cloning virtual machines.

Virtual machine disk

Storage of virtual machine data, including the operating system and application files. A disk is described by its storage parameters, the StorageClass and the size, and by a data source, which can be an image, another disk or a snapshot.

A disk is described by VirtualDisk. Without a data source an empty disk of the given size is created. With a source, Deckhouse Platform takes the size from the source and picks the storage class by it. Disk behaviour depends on the storage behind the StorageClass: on file system volumes the disk is created in the `qcow2` format, on block devices data is written directly, and the volume binding mode defines whether the disk is created right away or only after it is attached to a virtual machine.

For a ready disk only the size and the storage class can be changed. A disk is attached to a machine read-write as a block device. Its data can be captured in a snapshot or written to a file by disk data export.

For details, see Virtual machine disks.

Virtual machine image

The contents of a disk from which virtual machine disks are created. A project image is available only in the project or namespace where it was created, and a virtual machine accesses an attached image read-only.

An image is described by VirtualImage. Its source can be an HTTP server with the image file, a container image registry, a file uploaded from the command line, another image, a virtual machine disk or a disk snapshot. The downloaded image is stored either in DVCR or in a PVC. An image stored in a PVC can be used to create disks only in the same storage class.

To make an image available to every project of the cluster, an administrator creates a cluster image, ClusterVirtualImage. Images are attached to a virtual machine as block devices: read-only or as a CD-ROM for ISO images.

For details, see Virtual machine images.

Virtual machine lifecycle

The phases a virtual machine goes through from creation to deletion, shown in its status. Conditions explain why the machine is in its current phase.

In the `Pending` phase the machine waits for its dependent resources to become ready: disks, images, the VM class and the secret with the provisioning script. In the `Starting` phase Deckhouse Platform starts the machine on one of the nodes, in `Running` it works, in `Migrating` it moves to another node by live migration, and the `Terminating` phase is irreversible.

The `.status.conditions` block answers why the machine is in its phase: the conditions ending in `Ready` show what blocks the startup, `Running` reports the reason from the scheduler, `Migratable` shows whether the machine can be moved, `EvictionRequired` warns about eviction from a node. The run policy defines how the machine is kept running.

For details, see Creating a virtual machine.

Virtual machine pool

A set of identical virtual machines, replicas created from one template, whose number Deckhouse Platform maintains. A pool can be scaled through the scale subresource, HPA or KEDA. Available in commercial editions.

A pool is described by VirtualMachinePool. Its template matches a regular virtual machine specification, so a replica is no different from a manually created virtual machine. Replica disks are described once in disk templates with a reclaim policy, a size and a data source, and the template references them in the boot order. Shared images, such as one ISO for all replicas, are attached read-only.

Replicas are differentiated by provisioning, so the `Legacy` OS type is not supported in a pool.

For details, see Virtual machine pools.

Virtual machine snapshot

The captured state of a disk or of a whole virtual machine at a point in time, kept for recovery or cloning. A disk snapshot saves only the data of the selected disk, a VM snapshot includes the VM parameters and the state of all its disks.

A disk snapshot is described by VirtualDiskSnapshot and serves as a source for new disks and images. A virtual machine snapshot is described by VirtualMachineSnapshot and is the saved state of the machine at a certain moment. Attached images are not saved with it, so they have to be present in the cluster at recovery.

Snapshots are consistent or inconsistent, see consistent snapshot. A snapshot of a machine can also be the source of a virtual machine clone.

For details, see Snapshots of disks and virtual machines.

VM CPU topology

The distribution of the processor cores of a virtual machine across sockets. The user sets only the total number of cores, and Deckhouse Platform calculates the number of sockets.

The more cores, the more sockets they are divided into and the larger the step in which their number can be changed: up to 16 cores use one socket, up to 32 two, up to 64 four and up to 248 eight. The total number of cores has to be a multiple of the number of sockets. The topology matters for applications sensitive to the processor configuration.

The number of cores of a running machine can be changed within the current topology by live migration when the corresponding feature is enabled. A change that requires a different topology needs a restart.

For details, see Virtual machine CPU and memory.

VM disk data export

Writing the contents of a virtual machine disk or of its snapshot to a file so that the data can be moved outside the cluster.

Export requires the enabled `storage-volume-data-manager` module and, from the command line, the `d8` utility version 0.20.7 or later. The export is performed by the `d8 data export download` command. The disk must not be in use at the moment of export, so a machine it is attached to is stopped first. A snapshot can be exported instead of the disk.

A downloaded file is imported back into the cluster by uploading it as an image or as a disk.

For details, see Exporting virtual machine disk data.

VM eviction

Moving or restarting a virtual machine to free its node. It is triggered when the node is put into maintenance mode, and a machine that can be moved by live migration is not restarted.

The `EvictionRequired` condition appears when the node with the machine is put into node maintenance mode. A node that was only cordoned does not trigger it. The condition message says what will happen: a live migration without stopping the guest OS, a restart by Deckhouse Platform with the administrator’s permission, or waiting until the machine is restarted. Until the eviction starts, the condition is only a warning, because maintenance can be cancelled.

After a restart the machine starts on another suitable node, and without one it stays in the `Pending`

phase. A user can also evict a machine to another node with `d8 v evict` or a VirtualMachineOperation of the `Evict` type.

For details, see Creating a virtual machine.

VM migration policy

The policy that determines when a live migration of a virtual machine uses the AutoConverge mechanism, CPU throttling that guarantees the migration completes. Set by the liveMigrationPolicy parameter of the machine.

AutoConverge gradually lowers the CPU frequency of the source machine so that fewer dirty memory pages appear, and completes the live migration as soon as the transfer rate exceeds the memory change rate. It helps on a slow network at the cost of application performance in the machine.

The allowed values: `AlwaysSafe` (never throttle), `PreferSafe` (the default, no throttling, but a migration with `force=true` may throttle), `AlwaysForced` (always use AutoConverge) and `PreferForced` (use AutoConverge, but a migration with `force=false` runs without it).

For details, see Live migration of virtual machines.

VM placement rules

The rules that control on which node a virtual machine starts, from nodeSelector, node affinity, and affinity and anti-affinity to other machines and pods. Conditions are hard or soft.

A hard condition is mandatory: if no node satisfies it, the machine stays in the `Pending` phase. A soft condition is taken into account by the scheduler where possible. All rules, including the `nodeSelector` of the VM class, apply together. Nodes with taints need matching tolerations on the machine.

When the rules of a running machine change and its current node no longer meets them, commercial editions move the machine by live migration, and in Deckhouse Platform Open the change applies after a reboot. The placement rules of the machine, its disks and its class are also checked when a block device is attached. VM rebalancing restores the correct placement when the node stops meeting the rules.

For details, see Placing virtual machines on nodes.

VM pool disk reclaim policy

The reclaim policy of a disk template of a virtual machine pool. It sets what happens to a replica disk when the replica is removed from the pool.

The policy is set in the `virtualDiskTemplates` block of a virtual machine pool together with the size and the data source of the replica disk. With `Delete`, the default, the disk is deleted together with the replica. With `Retain`, the disk is kept and reused by the next replica.

For details, see Virtual machine pools.

VM provisioning

Initial configuration of the guest system of a virtual machine on its first boot by an initialization script, Cloud-Init for Linux or Sysprep for Windows.

The initialization script creates users, sets passwords and SSH keys, installs software, runs commands and enables services, for example the guest OS agent. The Cloud-Init configuration is written in YAML and starts with the `#cloud-config` header.

Replicas of a virtual machine pool are differentiated by provisioning, so operating systems without it, the `Legacy` OS type, are not supported in pools.

For details, see Initial provisioning and the guest OS agent.

VM rebalancing

Restoring an even distribution of virtual machines across nodes by moving them with live migration. Performed by the descheduler module.

Rebalancing evens out the load: when a node reserves more than 80% of CPU, some machines are moved to nodes that reserve less than 50%. It also restores correct placement when the current node no longer meets the machine requirements or the rules of mutual placement. With the `descheduler` module enabled, Deckhouse Platform creates a Descheduler named `virtualization` and selects in it only the machines that can be moved by live migration.

For details, see Virtual machine fault tolerance and balancing.

VM run policy

The policy that determines how Deckhouse Platform maintains the state of a virtual machine, set by the runPolicy parameter of the machine.

The allowed values: `AlwaysOnUnlessStoppedManually` (the default, the machine always runs and can be stopped only manually), `AlwaysOn` (the machine is started again even after a shutdown from the guest OS), `Manual` (the user manages the state) and `AlwaysOff` (the machine is always off and cannot be started).

The state of a virtual machine is managed by VirtualMachineOperation or by the `d8 v start`, `d8 v stop`, `d8 v restart` and `d8 v evict` commands, which create the same operation.

For details, see Connecting to a virtual machine.

Vulnerability database

Databases of known vulnerabilities used by image scanning with Trivy. The `operator-trivy` module downloads them from the registry every 6 hours. They take about 5 GB on the system nodes.

Scanning uses public vulnerability databases. The database images have fixed names and tags in the registry, so they can be copied into a private registry for an air-gapped cluster. The databases are stored on the system nodes. If a cluster has none, they are placed on a worker node. See also vulnerability scanning.

For details, see Disk layout and size.

Vulnerability scanning

Automatic detection of known vulnerabilities (CVE) and CIS compliance checks for the container images used by pods of the cluster, based on Trivy.

Deckhouse Platform scans the container images of all resources in namespaces labelled `security-scanning.deckhouse.io/enabled=""`. Without such labels only the `default` namespace is scanned. Scanning starts every 24 hours and when components with new images are deployed. The result of scanning each image is stored in the namespace of the scanned resource.

For details, see Container image vulnerability scanning.

Worker node

A node intended to run user applications. Its resources are dictated by the workload, with a part of them reserved for kubelet and system pods.

Worker nodes are present in every deployment option except the single-node cluster, where the only node takes their role. In a single-master cluster, some Deckhouse Platform components also run on worker nodes. The actual node size depends on the applications running on it.

For details, see the Resource requirements section of the “System requirements” page.

Workload

All projects together with the disk, network and other resources they use.

The workload runs in user namespaces. Running the workload in system namespaces is prohibited.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Frequently Asked Questions

The page contains frequently asked questions about configuring and using the Deckhouse Platform.

Common

How can I apply an update for a specific module immediately?

To apply an update for a specific module immediately, set the `modules.deckhouse.io/apply-now: "true"` annotation on the corresponding ModuleRelease resource.

This annotation applies the release immediately without waiting for the update window. The requirements from `spec.requirements` still apply. If they are not met, the release will not be applied.

Example of setting the annotation for the `console` module:

```
d8 k annotate mr console-v1.43.3 modules.deckhouse.io/apply-now="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module apply-now console v1.43.3
```

Example resource with the annotation set:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleRelease
metadata:
  name: console-v1.43.3
  annotations:
    modules.deckhouse.io/apply-now: "true"
...
```

How can I apply an update DP immediately, bypassing update windows, canary releases, and manual update mode?

To apply a Deckhouse Platform (DP) update immediately, add the annotation `release.deckhouse.io/apply-now: "true"` to the corresponding DeckhouseRelease resource.

This will bypass update windows, canary release settings, and the manual cluster update mode. The update will be applied immediately after the annotation is set.

Example command to set the annotation and skip update windows for version `v1.56.2`:

```
d8 k annotate deckhousereleases v1.56.2 release.deckhouse.io/apply-now="true"
```

Example of a resource with the annotation set:

```
apiVersion: deckhouse.io/v1alpha1
kind: DeckhouseRelease
metadata:
  annotations:
    release.deckhouse.io/apply-now: "true"
...
```

What to do if, with VPN enabled, the container with the installer cannot access the network?

If a VPN is installed on the computer from which the cluster bootstrap is performed, there may be a problem with the container with the installer (or graphical installer) accessing the network. Because of this, for example, the graphical installer may not be displayed in the browser.

The problem can be solved in one of the following ways:

- Disable VPN on your computer and restart the container with the installer.
- If you cannot disable VPN (for example, if the cluster bootstrap is running on a VPN network), use the `--network host` parameter when starting the container with the installer (**for Docker Desktop on Mac OS, the parameter is available starting with version 4.34.0**). This will allow the container to access the network.

  Example of launching a container with an installer with VPN enabled, using the `--network host`

parameter:

```
docker run --network host --pull=always -it -v "$PWD/config.yml:/config.yml" -v
"$HOME/.ssh/:/tmp/.ssh/" -v "$PWD/dhctl-tmp:/tmp/dhctl"
registry.deckhouse.ru/deckhouse/ce/install:early-access bash
```

Example of running the graphical installer with VPN enabled, using the `--network host` parameter:

```
docker run --network host --rm --pull always -v $HOME/.d8installer:$HOME/.d8installer -v
/var/run/docker.sock:/var/run/docker.sock -p 127.0.0.1:8080:8080
registry.deckhouse.ru/deckhouse/installer:latest -r $HOME/.d8installer
```

How do I change the release channel for a module?

A module can be built-in to DP or connected from a module source (defined using ModuleSource). Built-in modules have a common release cycle with DP and are updated together with DP. **The release channel of a built-in module always matches the DP release channel.** A module connected from a source has its own release cycle, which is independent of the DP release cycle. **The release channel of a module connected from a source can be changed.**

Below is the process of changing the release channel for a module connected from a source.

By default, the release channel for modules is inherited from the DP release channel (specified in the `releaseChannel` parameter of the `deckhouse` ModuleConfig). For more information about release channels, see the Release Channels section.

For modules from a source, the release channel is specified using ModuleUpdatePolicy, which is then *linked* to the module via the `updatePolicy` parameter in ModuleConfig.

To change the release channel for a module from a source, follow these steps:

1. Define the module update policy.

   Create a ModuleUpdatePolicy where you specify the release channel in the `releaseChannel` parameter.

   Example ModuleUpdatePolicy:

   ```
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: my-module-policy
   spec:
     releaseChannel: Alpha
     update:
       mode: Auto
   ```

   If necessary, change the update mode and configure update windows.

   Ensure that the policy has been created:

   ```
   d8 k get mup my-module-policy
   ```

   Output example:

   ```
   NAME               RELEASE CHANNEL   UPDATE MODE
   my-module-policy   Alpha             AutoPatch
   ```

2. Link the update policy to the module.

   Specify the name of the created update policy in the updatePolicy parameter of the corresponding module's ModuleConfig.

   To edit the ModuleConfig, use the command (specify the module name):

   ```
   d8 k edit mc my-module
   ```

Example ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: my-module
spec:
  enabled: true
  # ModuleUpdatePolicy name.
  updatePolicy: my-module-policy
```

When you change the module's release channel, its version will change according to the configured update mode.

To view the current release channel of the module and other information about the module's state in the cluster, use the corresponding Module object.

Example command to get information about the module:

```
d8 k get module my-module -o yaml
```

The update policy used will be specified in the `properties.updatePolicy` field, and the current release channel in the `properties.releaseChannel` field. Example output:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  name: my-module
  # ...
properties:
  # ...
  releaseChannel: Alpha # Module release channel.
  updatePolicy: my-module-policy # Module update policy.
  version: v1.16.10  # Module version.
  # ...
```

How do I configure an external module to update to patch versions only?

By default, if a module has no dedicated update policy, the update mode and windows are inherited from the DP settings.

If DP is set to the `AutoPatch` mode, the external module will also automatically receive only patch versions within the current minor version. Moving to a new minor version will require manual approval. In this case, no additional configuration is needed.

For more information about DP update modes, see Configuring updates.

If you need to manage the module update mode independently of DP, create a ModuleUpdatePolicy with `update.mode: AutoPatch` and link it to the module via the `updatePolicy` parameter in ModuleConfig:

1. Create an update policy.

   Example ModuleUpdatePolicy:

   ```
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: my-update-policy
   spec:
     releaseChannel: Stable
     update:
       mode: AutoPatch
   ```

   Ensure that the policy has been created:

```
d8 k get mup my-update-policy
```

2. Link the policy to the module.

   Specify the policy name in the updatePolicy parameter of the module's ModuleConfig:

   ```
   d8 k edit mc module
   ```

   Example ModuleConfig:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: module
   spec:
     enabled: true
     updatePolicy: my-update-policy
   ```

In the `AutoPatch` mode, module patch versions (for example, from `v1.16.1` to `v1.16.2`) are applied automatically, taking update windows into account if they are configured. To move to a new minor version (for example, from `v1.16.*` to `v1.17.*`), approve the corresponding ModuleRelease:

```
d8 k annotate mr module-v1.17.0 modules.deckhouse.io/approved="true"
```

Or using the `d8` CLI:

```
d8 system module approve module v1.17.0
```

What should I do if the module image did not download and the module did not reinstall?

In some cases, there may be a problem with automatically downloading the image and reinstalling the module. These cases include:

- Damage to the file system or other problems that have rendered the module image invalid.
- Switching to a different registry.
- Switching from one DP edition to another.

In this case, the module may be in the `Ready` state. The error occurs in the module's pods. To find the problematic pod, use the command:

```
d8 k -n d8-<module-name> get pods
```

A problematic pod will have a status other than `Running`.

To view information about a pod, use the command:

```
d8 k -n d8-<module-name> describe pod <pod-name>
```

Example of an error message in the pod when there is a problem with downloading the image and reinstalling the module:

```
Failed to pull image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0
a7c968a015dd8d18bb5d54645ddb040ddab1ef4": rpc error: code = NotFound desc = failed
to pull and unpack image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0
a7c968a015dd8d18bb5d54645ddb040ddab1ef4": failed to resolve reference ...
```

To download the image and reinstall the module that caused the problem:

1. Get a list of module releases:

```
d8 k get mr -l module=my-module
```

Output example:

```
NAME                PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
my-module-v3.7.4    Superseded                   5d23h
my-module-v3.7.5    Deployed                     5d23h
```

Find the module release deployed in the cluster in the list (it should have the status `Deployed`).

2. Add the annotation `modules.deckhouse.io/reinstall=true` to the expanded release:

```
d8 k annotate mr my-module-v3.7.5 modules.deckhouse.io/reinstall=true
```

After adding the annotation, the module image is re-downloaded from the registry, the module is validated with the current settings from `ModuleConfig`, and installed in the cluster. After successful reinstallation, the annotation is automatically removed from `ModuleRelease`.

To verify that the module has been successfully reinstalled and all module pods are working, use the command:

```
d8 k -n d8-<module-name> get pods
```

All pods in the module must have the status `Running`. Example:

```
NAME                                READY   STATUS    RESTARTS   AGE
backend-567d6c6cdc-g5qgt            1/1     Running   0          2d2h
frontend-7c8b567759-h8jdf           1/1     Running   0          2d2h
observability-gw-86cf75f5d6-7xljh   1/1     Running   0          2d2h
```

Cluster and Infrastructure

What to do if you encounter problems updating Kubernetes components on cluster nodes, synchronizing nodes, or applying NodeGroup Configuration?

If Kubernetes components are not updated on the cluster node, the NodeGroup configuration is not applied, and not all NodeGroup nodes are synchronized (have the `UPTODATE` status), perform the following steps:

1. Check the bashible logs on the node where the problems are occurring. The bashible mechanism is used to keep cluster nodes up to date. It is started by the `bashible.timer` timer at regular intervals as a service on the cluster nodes. This involves restarting, synchronizing scripts, and executing them (if necessary).

   To check bashible logs, use the command:

   ```
   journalctl -u bashible
   ```

   If the response contains the message `Configuration is in sync, nothing to do`, the node is synchronized and there are no problems. The absence of this message or the presence of errors indicates a problem.

2. Check the synchronization status of cluster nodes using the command:

   ```
   d8 k get ng
   ```

   The number of nodes in the `UPTODATE` state must match the total number of nodes in each group.

   Example output:

```
NAME       TYPE     READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
  STANDBY   STATUS   AGE     SYNCED
frontend   Static   1       1       1
                    118d    True
master     Static   3       3       3
                    118d    True
system     Static   2       2       2
                    118d    True
worker     Static   2       2       2
                    118d    True
```

How to migrate node groups to Cluster API (CAPI)?

Deckhouse Platform is migrating CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI).

At the moment, migration from MCM to CAPI is supported for the following cloud providers:

- Yandex Cloud
- OpenStack
- VMware vSphere

After CAPI support is introduced for a cloud provider, existing CloudEphemeral node groups continue to use MCM (`status.engine: MCM`). New node groups use CAPI by default (`status.engine: CAPI`).

To check which management mechanism is used for a node group, run the following command:

```
d8 k get nodegroup -o custom-columns=NAME:.metadata.name,ENGINE:.status.engine
```

To migrate a node group from MCM to CAPI:

1. Create a new NodeGroup of the CloudEphemeral type with the required configuration. Do not set the `node.deckhouse.io/use-mcm` annotation — otherwise the group will remain on MCM.
2. Make sure the new group is managed by CAPI (`status.engine: CAPI`):

   ```
   d8 k get nodegroup <NODE_GROUP_NAME> -o custom-columns=NAME:.metadata.name,ENGINE:.status.engine
   ```

3. Wait until the nodes in the new group become `Ready`:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<NODE_GROUP_NAME>
   ```

4. Move workloads from the old group to the new one.

   For example, update application `nodeSelector` and `tolerations` so that pods are scheduled onto nodes of the new group, or use another workload placement method adopted in your infrastructure. For details on allocating nodes for workloads, see Allocating nodes for specific workloads.

5. Make sure the migrated workloads are running successfully on the new group nodes and that no required pods remain on the old group nodes.
6. Delete the old NodeGroup.

> ⚠ Deleting a NodeGroup deletes all nodes in that group. Before deleting the old group, make sure the required workloads have been moved on the new group nodes and are running correctly.

If necessary, you can force a node group to be created using MCM. To do this, add the `node.deckhouse.io/use-mcm` annotation before creating the NodeGroup (or before recreating it):

```
d8 k annotate nodegroup <NODE_GROUP_NAME> node.deckhouse.io/use-mcm="true"
```

For example:

```
d8 k annotate nodegroup worker node.deckhouse.io/use-mcm="true"
```

⚠ The `node.deckhouse.io/use-mcm` annotation is a temporary workaround and its use is not recommended. Migrating node groups to CAPI is the preferred approach.

What to do if problems adding a node to the cluster via Cluster API Provider Static are present?

If, when adding a node to the cluster via Cluster API Provider Static (CAPS), it remains in `Pending` or `Bootstrapping` status, perform the following steps:

1. Verify that the access keys specified in the SSHCredentials resource are correct. Ensure that the username and SSH key specified in SSHCredentials are correct.
2. On the node where the problem occurred, check that the public key corresponding to the private key from SSHCredentials is present in `authorized_keys`. Example command for checking:

   ```
   cat ~/.ssh/authorized_keys
   ```

3. Check the number of nodes specified in NodeGroup, which should include the node being added. Make sure that the maximum number of nodes is not exceeded.
4. Check the status of the `bashible.service` on the node that caused the problem:

   ```
   systemctl status bashible.service
   ```

   It must have the status `active (running)`. If the service has the status `inactive` or `failed`, the service has not started. This indicates a problem with the configuration process.
5. If the steps above did not resolve the issue, remove the problematic node and its StaticInstance resource from the cluster so that the system will attempt to recreate them. To do this:
   - Get a list of nodes and locate the problematic one:

     ```
     d8 k get nodes
     ```

   - Find the corresponding StaticInstance resource:

     ```
     kubectl get staticinstances -n <namespace-name>
     ```

   - Remove the problematic node:

     ```
     kubectl delete node <node-name>
     ```

   - Remove the corresponding StaticInstance resource:

     ```
     kubectl delete staticinstances -n <namespace-name> <static-instance-name>
     ```

How do I replace the OS image in a cloud cluster?

Deckhouse Platform creates `CloudEphemeral` and `CloudPermanent` nodes, including master nodes, based on an operating system (OS) image specified in the node group configuration. To move the nodes to a new image, specify it in this configuration instead of the previous one. Deckhouse Platform recreates the `CloudEphemeral` nodes from the new image automatically. The `CloudPermanent` nodes and master nodes are recreated when you apply the configuration with the `dhctl converge` command.

For `CloudStatic` nodes, the administrator creates and maintains the virtual machines. To replace the image of such a node, prepare a virtual machine with the required OS and connect it using the instructions for adding CloudStatic nodes.

The resource where the image is specified depends on the node type and the cloud provider configuration method:

Node type	Provider configuration method	Where to change the image
`CloudEphemeral`	Through ModuleConfig or `<PROVIDER>ClusterConfiguration`	In the `<PROVIDER>InstanceClass` resource referenced by the node group
`CloudPermanent` and master nodes	Through ModuleConfig	In the `<PROVIDER>InstanceClass` resource referenced by the node group
`CloudPermanent` and master nodes	Through `<PROVIDER>ClusterConfiguration`	In the `instanceClass` section of the required node group in that resource

> **i** Migration from `<PROVIDER>ClusterConfiguration` to the new ModuleConfig-based configuration model is supported only for built-in virtualization (DVP), Yandex Cloud, and zVirt.

The image parameters depend on the resource whose configuration you edit. Select the appropriate tab.

- InstanceClass
- ClusterConfiguration

For `CloudEphemeral` nodes, use the instance class parameters with either provider configuration method. With a ModuleConfig-based configuration, these parameters also define the image for `CloudPermanent` nodes and master nodes.

Cloud provider	Parameter in the `spec` section	Instance class resource
Amazon Web Services	`ami`	AWSInstanceClass
Microsoft Azure	`urn`	AzureInstanceClass
Google Cloud Platform	`image`	GCPInstanceClass
Yandex Cloud	`imageID`	YandexInstanceClass
OpenStack	`imageName`	OpenStackInstanceClass
Huawei Cloud	`imageName`	HuaweiCloudInstanceClass
Basis Dynamix	`imageName`	DynamixInstanceClass
VMware Cloud Director	`template`	VCDInstanceClass
VMware vSphere	`template`	VsphereInstanceClass
zVirt	`template`	ZvirtInstanceClass
Built-in virtualization	`rootDisk.image`	DVPInstanceClass

With a `<PROVIDER>ClusterConfiguration`-based provider configuration, the master node image is set in `masterNodeGroup.instanceClass`, and the image for other `CloudPermanent` nodes is set in `nodeGroups[].instanceClass`.

Cloud provider	Master nodes	`CloudPermanent` worker nodes
Amazon Web Services	`masterNodeGroup.instanceClass.ami`	`nodeGroups[].instanceClass.ami`
Microsoft Azure	`masterNodeGroup.instanceClass.urn`	`nodeGroups[].instanceClass.urn`
Google Cloud Platform	`masterNodeGroup.instanceClass.image`	`nodeGroups[].instanceClass.image`
Yandex Cloud	`masterNodeGroup.instanceClass.imageID`	`nodeGroups[].instanceClass.imageID`

Cloud provider	Master nodes	`CloudPermanent` worker nodes
OpenStack	`masterNodeGroup.instanceClass.imageName`	`nodeGroups[].instanceClass.imageName`
Huawei Cloud	`masterNodeGroup.instanceClass.imageName`	`nodeGroups[].instanceClass.imageName`
Basis Dynamix	`masterNodeGroup.instanceClass.imageName`	`nodeGroups[].instanceClass.imageName`
VMware Cloud Director	`masterNodeGroup.instanceClass.template`	`nodeGroups[].instanceClass.template`
VMware vSphere	`masterNodeGroup.instanceClass.template`	`nodeGroups[].instanceClass.template`
zVirt	`masterNodeGroup.instanceClass.template`	`nodeGroups[].instanceClass.template`
Built-in virtualization	`masterNodeGroup.instanceClass.rootDisk.image`	`nodeGroups[].instanceClass.rootDisk.image`

With cloud providers other than VMware Cloud Director, VMware vSphere, zVirt, and built-in virtualization, the parameter value is a string with the OS image name or ID. In VMware Cloud Director, VMware vSphere, and zVirt, the `template` parameter sets a virtual machine template prepared based on the OS image. To move the nodes of these providers to a new image, prepare a new virtual machine template based on it.

> **i** In built-in virtualization, the image is specified as an object. In the `rootDisk.image.kind` parameter, specify the type of the resource with the image (`ClusterVirtualImage`, `VirtualImage`, or `VirtualDisk`), and in the `rootDisk.image.name` parameter, specify the name of this resource.

Example of the image parameter in built-in virtualization

```
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: worker
spec:
  rootDisk:
    # The other instance class parameters are not shown.
    image:
      kind: ClusterVirtualImage
      name: ubuntu-24-04-20260204
```

Requirements for the new image

The OS image or the virtual machine template prepared from it must meet the requirements of the selected cloud provider module.

Before the replacement, make sure that the new image or virtual machine template meets the following general requirements:

- The image parameter value differs from the previous one. If the image or the template is set by name, include the version or the date in the name, for example `ubuntu-24-04-20260204`.
- The image or the template prepared based on it exists in the cloud by the time the nodes are recreated.
- The image contains `cloud-init` and an operating system supported by Deckhouse Platform. The list of supported systems is given in “Supported OS versions”.

For some providers, preparing the image or template involves additional steps. Follow the instructions in the corresponding module documentation:

- For OpenStack, refer to “How do I create an image in OpenStack?”.
- For Basis Dynamix, refer to “Prepare an operating system image”.
- For VMware Cloud Director, refer to “Virtual machine template”.

- For VMware vSphere, refer to “VM image requirements”, which includes instructions for preparing the image.
- For zVirt, refer to “Prepare an operating system image” and “Prepare a virtual machine template”.

⚠ Deckhouse Platform compares the parameter values in the configuration, not the image content in the cloud. If you replace the image or the template in the cloud keeping its previous name, the nodes are not recreated.

Changing the image in the configuration

Create a new image in the cloud that meets the requirements above. In VMware Cloud Director, VMware vSphere, and zVirt, prepare a new virtual machine template based on it. Then select the cloud provider configuration method.

⚠ In Yandex Cloud, `CloudEphemeral` groups whose instance classes do not specify `imageID` use the master node image by default. Changing the master node image does not recreate the existing nodes in these groups. New nodes may use the new image, so a group can contain nodes with different images.

To keep the previous group image, explicitly set its `imageID` in YandexInstanceClass. To recreate the entire group from the new image, explicitly set the new `imageID`.

For DVP, Yandex Cloud, and zVirt, check the master node group configuration before selecting a tab:

```
d8 k get nodegroup master -o yaml
```

If `spec.cloudInstances.classReference.kind` and `spec.cloudInstances.classReference.name` are set and migration is complete, select the “Through ModuleConfig” tab. Make sure the migration alert for your provider is not firing:

- For DVP, this is `D8CloudProviderDVPMigrationPending`.
- For Yandex Cloud, this is `D8CloudProviderYandexCloudMigrationPending`.
- For zVirt, this is `D8CloudProviderZvirtMigrationPending`.

If the alert is firing, complete the migration first. If the master node group has no `classReference`, or the cluster uses another provider, select the “Through ClusterConfiguration” tab.

i After completing the migration to ModuleConfig, change the image in the instance class referenced by the required node group.

The cluster may retain the `d8-provider-cluster-configuration` Secret with the old provider configuration. The `d8 system edit provider-cluster-configuration` command lets you edit this configuration, but the platform no longer applies image changes from it.

Select the appropriate tab depending on how the cloud provider is configured:

- Through ModuleConfig
- Through ClusterConfiguration

With a ModuleConfig-based provider configuration, the image for `CloudEphemeral` and `CloudPermanent` nodes, including master nodes, is specified in an instance class. To replace the image, edit the instance class referenced by the required node group:

1. Open the node group configuration:

   ```
   d8 k get nodegroup <NODE_GROUP_NAME> -o yaml
   ```

   Replace `<NODE_GROUP_NAME>` with the node group name. For master nodes, use `master`.

   Check `spec.nodeType` for the node type and `spec.cloudInstances.classReference.kind` and `spec.cloudInstances.classReference.name` for the instance class kind and name. If multiple groups use the instance class, changing the image affects all of them.

2. Specify the new image or virtual machine template in the instance class of the node group and save the file:

```
d8 k edit <PROVIDER>instanceclass <INSTANCE_CLASS_NAME>
```

Replace `<PROVIDER>` with the cloud provider name in lowercase, for example `yandex`, and `<INSTANCE_CLASS_NAME>` with the instance class name.

An example for Yandex Cloud:

```
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: worker
spec:
  cores: 4
  memory: 8192
  imageID: fd83ergat2e815oohe7o
```

Example of the master node configuration in built-in virtualization

```
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: master
spec:
  rootDisk:
    # The other instance class parameters are not shown.
    image:
      kind: ClusterVirtualImage
      name: ubuntu-24-04-20260204
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: master
    minPerZone: 1
    maxPerZone: 1
```

After changing the instance class, proceed to applying the changes.

With a `<PROVIDER>ClusterConfiguration`-based provider configuration, the resource to edit depends on the node type. Select the node type of the group.

- CloudEphemeral
- CloudPermanent
- Master nodes

For `CloudEphemeral` nodes, the image is specified in the instance class referenced by the node group:

1. Check `spec.cloudInstances.classReference.kind` and `spec.cloudInstances.classReference.name` in the NodeGroup resource for the instance class kind and name:

   ```
   d8 k get nodegroup <NODE_GROUP_NAME> -o yaml
   ```

   Replace `<NODE_GROUP_NAME>` with the node group name.

2. Open the instance class and change the image parameter listed in the table above:

   ```
   d8 k edit <PROVIDER>instanceclass <INSTANCE_CLASS_NAME>
   ```

Replace `<PROVIDER>` with the cloud provider name in lowercase, for example `yandex`, and `<INSTANCE_CLASS_NAME>` with the instance class name.

For `CloudPermanent` nodes, change the image in the provider configuration:

1. Open the cloud provider configuration:

   ```
   d8 system edit provider-cluster-configuration
   ```

2. Change the image parameter in the `instanceClass` section of the required group in the `nodeGroups` list and save the file. An example for Yandex Cloud:

   ```
   nodeGroups:
   - name: front
     replicas: 2
     instanceClass:
       cores: 4
       memory: 8192
       imageID: fd83ergat2e815oohe7o
   ```

For master nodes, change the image in the provider configuration:

1. Open the cloud provider configuration:

   ```
   d8 system edit provider-cluster-configuration
   ```

2. Change the image parameter in the `masterNodeGroup.instanceClass` section and save the file.

After saving the configuration, apply the changes according to the node type.

Applying the changes

The way to apply the changes depends on the node type and is the same for both provider configuration methods. Select the appropriate tab.

- CloudEphemeral
- CloudPermanent and master nodes

Deckhouse Platform automatically recreates `CloudEphemeral` nodes after the image in the instance class changes. To check that the replacement is complete, follow these steps:

1. Wait until Deckhouse Platform recreates the nodes of the group:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<NODE_GROUP_NAME>
   ```

   Replace `<NODE_GROUP_NAME>` with the node group name.

   During the replacement, the list may contain both the previous and the new nodes. The new nodes have a smaller value in the `AGE` column. An example of the output during the replacement:

   ```
   NAME                                        STATUS     ROLES    AGE     VERSION
   cloud-demo-worker-92f9049e-gm2lw-6hkm2      NotReady   worker   37s     v1.34.1
   cloud-demo-worker-92f9049e-xn29v-66rm9      Ready      worker   8m21s   v1.34.1
   ```

   The replacement is complete when only the new nodes in the `Ready` status remain in the list.

To apply the new image to `CloudPermanent` nodes and master nodes, run `dhctl converge`. The command reads the current configuration from the cluster, including ModuleConfig, NodeGroup, and instance classes when the provider uses a ModuleConfig-based configuration.

Before replacing master nodes, back up etcd. Also back up the `/etc/kubernetes` directory and copy both backups outside the cluster. Master nodes are recreated one at a time. In a single-master cluster, the command may offer a temporary transition to three master nodes and a return to one master node after the replacement.

1. On the local machine, get the Deckhouse Platform edition and version in the cluster:

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
'[:lower:]')
```

2. On the local machine, run the Deckhouse Platform installer container of the same edition and version:

   ```
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

   If the cluster uses a different container registry, specify its address instead of `registry.deckhouse.io`.

3. In the installer container, check which changes will be made in the cloud:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
     --ssh-user=<USERNAME> --ssh-host=<MASTER_IP>
   ```

   Replace `<SSH_KEY_FILENAME>` with the name of the private SSH key file, `<USERNAME>` with the user for connecting to the master node, and `<MASTER_IP>` with the master node address.

   When replacing master nodes, specify all master node addresses by repeating `--ssh-host` for each address.

   In the `dhctl terraform check` output, verify that the changes to nodes and other cloud resources match your expectations. In the final `nodes` list, nodes that require recreation have the `destructively_changed` status. If multiple groups use the same instance class, check the changes for each group.

4. In the installer container, apply the changes using the same SSH connection parameters as in the check:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
     --ssh-user=<USERNAME> --ssh-host=<MASTER_IP>
   ```

   The command displays the plan of changes and asks for the `Do you want to CHANGE objects state in the cloud?` confirmation before replacing each node. To replace the node, enter `y`. The command also asks you to confirm that the master node name matches its address.

   Before confirming the replacement of the next master node, check etcd and the control plane using the master node verification procedure.

5. Check that the nodes of the group are recreated:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<NODE_GROUP_NAME>
   ```

   Replace `<NODE_GROUP_NAME>` with the node group name. For master nodes, use `master`. Wait until the new nodes reach the `Ready` status.

How to change the instance type for nodes with the CloudPermanent type?

To change the instance type for nodes with the CloudPermanent type, follow these steps:

1. Make a backup of etcd and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty. To view the status of all Deckhouse job queues, run the following command:

   ```
   d8 s queue list
   ```

   Example output (queues are empty):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

5. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
'[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

6. **In the installer container**, run the following command to check the state before working:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
HOST>
```

   The command output should indicate that Terraform found no inconsistencies and no changes are required.

7. **In the installer container**, run the command to edit the cluster configuration (specify the addresses of all master nodes in the `--ssh-host` parameter):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-
keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
HOST>
```

8. Edit the `instanceClass` parameter of the desired node group by changing the instance type and save the changes. Example settings for the `masterNodeGroup` of the Yandex Cloud provider:

```
masterNodeGroup:
 replicas: 3  # required number of master nodes
 instanceClass:
   cores: 4     # change the number of CPUs
   memory: 8192  # change the memory size (in MB)
   # other instance parameters...
   externalIPAddresses:
   - "Auto"     # for each master node
   - "Auto"
   - "Auto"
```

9. **In the installer container**, run the following command to perform nodes upgrade:

   You should read carefully what converge is going to do when it asks for approval.

   When the command is executed, the nodes will be replaced by new nodes with confirmation on each node. The replacement will be performed one by one in reverse order (2,1,0).

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME>
\
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-
HOST>
```

   Repeat the steps below (Sec. 9-12) for each master node one by one, starting with the node with the highest number (suffix 2) and ending with the node with the lowest number (suffix 0).

10. **On the newly created node**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

    ```
    journalctl -fu bashible.service
    ```

11. Make sure the node is listed as an etcd cluster member:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

12. Make sure `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
    ```

13. Proceed to update the next node (repeat the steps above).

How to migrate a cloud provider to ModuleConfig-based configuration?

Migration to the new ModuleConfig-based configuration model is supported only for built-in virtualization (DVP), Yandex Cloud, and zVirt. If the cluster uses DVPClusterConfiguration, YandexClusterConfiguration, or ZvirtClusterConfiguration, migrate its configuration using the procedure below.

Deckhouse Platform is transitioning from a single `<PROVIDER>ClusterConfiguration` resource to a model where the cloud provider configuration is split across four separate resources:

- ModuleConfig of the `cloud-provider-<PROVIDER>` module: Provider and layout settings
- Secret with credentials of the `cloud-provider.deckhouse.io/credentials` type: Access to the cloud API
- Provider InstanceClass (for example, DVPInstanceClass): Virtual machine parameters
- NodeGroup: Node group configuration

> ⚠ Migration is mandatory. This is not an optional step: support for `<PROVIDER>ClusterConfiguration` will be removed. Until the migration is completed, Deckhouse Platform updates may be blocked.

The migration is safe. Applying the resources automatically prepared by Deckhouse Platform **does not cause nodes to be recreated**. However, explicit administrator action is required, as the resources have to be reviewed and applied manually.

How to migrate

1. Wait for the migration alert. Each provider has its own, for example, `D8CloudProviderDVPMigrationPending` for DVP, `D8CloudProviderYandexCloudMigrationPending` for Yandex Cloud, and `D8CloudProviderZvirtMigrationPending` for zVirt. The alert means the module has detected the legacy configuration and prepared manifests for the transition.
2. Review the prepared resources in the `d8-migration-resources` Secret in the `d8-cloud-provider-<PROVIDER>` namespace:

    ```
    d8 k -n d8-cloud-provider-<PROVIDER> get secret d8-migration-resources -o
    jsonpath='{.data.resources\.yaml}' | base64 -d
    ```

    Example for built-in virtualization:

```
d8 k -n d8-cloud-provider-dvp get secret d8-migration-resources -o
jsonpath='{.data.resources\.yaml}' | base64 -d
```

3. Verify that the manifests match the expected cluster configuration (ModuleConfig, credentials Secret, InstanceClass, and NodeGroup).
4. Apply the resources:

   ```
   d8 k -n d8-cloud-provider-<PROVIDER> get secret d8-migration-resources -o
   jsonpath='{.data.resources\.yaml}' | base64 -d | d8 k apply -f -
   ```

   Example for built-in virtualization:

   ```
   d8 k -n d8-cloud-provider-dvp get secret d8-migration-resources -o
   jsonpath='{.data.resources\.yaml}' | base64 -d | d8 k apply -f -
   ```

5. Wait until the alert disappears. The module clears it automatically once it detects that all required resources are in place.

How to delete the legacy secret

After the migration of a cluster deployed in built-in virtualization, Yandex Cloud, or zVirt, the provider settings are defined in ModuleConfig, and you can manually delete the legacy `d8-provider-cluster-configuration` Secret in the `kube-system` namespace.

To delete the Secret, follow these steps:

1. Make sure the migration is complete and its alert no longer fires. For built-in virtualization, it is the `D8CloudProviderDVPMigrationPending` alert, for Yandex Cloud, the `D8CloudProviderYandexCloudMigrationPending` alert, and for zVirt, the `D8CloudProviderZvirtMigrationPending` alert. If you delete the Secret before the migration is complete, the cloud provider module stops working correctly.
2. Check the number of nodes in node groups of the `CloudPermanent` type:

   ```
   d8 k get nodegroups \
     -o custom-
   columns='NAME:.metadata.name,TYPE:.spec.nodeType,MIN_PER_ZONE:.spec.cloudInstance
   s.minPerZone'
   ```

   After the Secret is deleted, `dhctl converge` takes the number of nodes in these groups from the `spec.cloudInstances.minPerZone` parameter. If the parameter is not set for a node group of the `CloudPermanent` type, `dhctl converge` fails.
3. Make sure the cloud infrastructure data, such as network IDs and zones, is stored in a separate Secret. The command must print a non-empty value:

   ```
   d8 k -n d8-cloud-provider-<PROVIDER> get secret d8-candi-cloud-provider-discovery-data \
     -o jsonpath='{.data.cloud-provider-discovery-data\.json}'
   ```

   Replace `<PROVIDER>` with `dvp` for built-in virtualization, `yandex` for Yandex Cloud, or `zvirt` for zVirt.

   If the command prints nothing or the Secret is not found, the data is stored only in the legacy Secret. If you delete it without a copy of the data, the `terraform-manager` module is disabled. The `cloud-provider-yandex` module also loses the IDs of the network, subnets, and route table that the platform created in Yandex Cloud. Copy the data to a separate Secret and delete the temporary file:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration \
  -o jsonpath='{.data.cloud-provider-discovery-data\.json}' \
  | base64 -d > /tmp/discovery-data.json

d8 k -n d8-cloud-provider-<PROVIDER> create secret generic d8-candi-cloud-provider-discovery-data \
  --from-file=cloud-provider-discovery-data.json=/tmp/discovery-data.json \
  --dry-run=client -o yaml \
  | d8 k apply -f -

rm /tmp/discovery-data.json
```

Then repeat the check and make sure the command prints a non-empty value.

4. Back up the Secret. The Secret contains cloud credentials, so store the file securely:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration \
  -o yaml > d8-provider-cluster-configuration-backup.yaml
```

5. Allow deletion of the Secret. The Secret is protected by a webhook that rejects deletion without the `deckhouse.io/allow-delete` annotation:

```
d8 k -n kube-system annotate secret d8-provider-cluster-configuration \
  deckhouse.io/allow-delete=true
```

6. Delete the Secret:

```
d8 k -n kube-system delete secret d8-provider-cluster-configuration
```

What to do if it takes a long time to switch to custom nodes in lower-priority groups?

When using multiple node groups with different priorities in a cloud cluster (the `spec.cloudInstances.priority` parameter), switching to nodes from groups with lower priority can take a long time. An example of this scenario is when groups of preemptible nodes (spot, etc.) are set to the highest priority, and if such instances are unavailable, ordering nodes from other groups takes a very long time.

When provisioning nodes, the cluster autoscaler sequentially selects the group with the highest priority that is **not in a backoff state**. A backoff is a temporary lock on a group following a failed attempt to provision a node (for example, due to a lack of instances of the required type in the cloud).

How it works when using multiple node groups:

1. The cluster autoscaler attempts to provision a node in the group with the highest priority.
2. If the node is not provisioned within the time specified by the `max-node-provision-time` parameter, the attempt is considered a failure.
3. The group is marked as `failed` and blocked for the duration specified in the `initial-node-group-backoff-duration` parameter.
4. If the same group fails again, the lockout time is doubled, but does not exceed the time specified in the `max-node-group-backoff-duration` parameter.
5. The cluster autoscaler selects the next highest-priority group that is not locked out.

Here:

Parameter	Default value	Description
`initial-node-group-backoff-duration`	5 minutes	Initial duration of the group lockout after the first failure. Doubles after each failed node ordering attempt
`max-node-group-backoff-duration`	30 minutes	Maximum duration of the group lockout

Parameter	Default value	Description
`max-node-provision-time`	15 minutes	The time after which the cluster autoscaler considers a node provisioning attempt to have failed

If these parameters are set to their default values, switching to node ordering in lower-priority groups may take a considerable amount of time (for more details, see this example).

These settings can be adjusted to speed up the switching of nodes in lower-priority groups. To set the desired values for the `initial-node-group-backoff-duration`, `max-node-group-backoff-duration`, and `max-node-provision-time` parameters, edit the `cluster-autoscaler` Deployment object in the `d8-cloud-instance-manager` namespace. The parameters are specified in the `args` field of the `cluster-autoscaler` container. Example:

```
...
    containers:
     - name: cluster-autoscaler
       args:
       - --initial-node-group-backoff-duration=1m # The initial duration of the group
lockout after the first failure has been reduced.
       - --max-node-provision-time=5m # The time after which the cluster autoscaler
considers a node request to have failed has been reduced.
...
```

If you encounter issues with node ordering in a cluster with a single node group

If cloud cluster uses only one node group (for example, exclusively for ordering spot or preemptible nodes) and you encounter issues with ordering such nodes during automatic scaling, follow these steps:

1. Create one or more InstanceClass objects to be used when creating instances in the cluster. In the InstanceClass, specify node types that differ from those listed in the group mentioned above.

   > **i** In DP, InstanceClass objects vary depending on the provider. Examples: AWSInstanceClass, AzureInstanceClass, YandexInstanceClass.

2. Create one or more new node groups (NodeGroup). In the `spec.cloudInstances.classReference` parameter, specify the InstanceClass objects created in the previous step. In the `spec.cloudInstances.priority` parameter, set the priority of the node group.
3. If necessary, modify the `max-node-provision-time`, `max-node-group-backoff-duration`, and `initial-node-group-backoff-duration` parameters to speed up switching to lower-priority groups.

Example of the node ordering process with default settings

Suppose there are 3 groups of nodes with different priorities:

- **Group A** (priority 50)
- **Group B** (priority 30)
- **Group C** (priority 0)

In this case, the node ordering process might be as follows:

Time	Event
10:00	The cluster autoscaler attempts to order a node in **A**
10:15	The attempt to provision a node in **A** within 15 minutes (`max-node-provision-time`) fails → **A** blocked for 5 minutes (`initial-node-group-backoff-duration`)
10:15:xx	The cluster autoscaler attempts to order a node in **B** (next in priority, not blocked)

Time	Event
10:30	The attempt to provision a node in **B** within 15 minutes (`max-node-provision-time`) fails → **B** blocked for 5 minutes (`initial-node-group-backoff-duration`)
10:30:xx	The cluster autoscaler attempts to order a node in **A** again, as the lockout period (the default value for `initial-node-group-backoff-duration` is 5 minutes) has ended
10:45	The attempt to order a node in **A** fails again → **A** is blocked for 10 minutes (`initial-node-group-backoff-duration` for **A** is doubled)
…	And so on, until the lockout time for **A** and **B** exceeds 15 minutes (`max-node-provision-time`). In this case, the queue may not reach group **C** for about 1.5 hours

What to do if the resources generated by an object are not created after adding the object?

If the required resources do not appear after creating an object in the system (for example, dexAuthenticator), follow these steps:

1. Check if there are any critical alerts in the cluster that may block the creation of the required objects. To do this, use the command:

   ```
   d8 k get clusteralerts.deckhouse.io
   ```

   Example output:

   ```
   NAME               ALERT                                          SEVERITY
   AGE   LAST RECEIVED   STATUS
   012f602592aa7a91   K8SSchedulerTargetDown                         3
   16h   54s             firing
   0836dc893d5ecc65   KubernetesDeploymentReplicasUnavailable        5
   15h   62s             firing
   08742f87d62d0063   NTPDaemonOnNodeDoesNotSynchronizeTime          5
   16h   46s             firing
   172cfd38d2f7fd19   D8DeckhouseQueueIsHung                         7
   12h   66s             firing
   1c5705daf731f5cf   D8StrongholdNoActiveNodes                      3
   16h   55s             firing
   1d2c2f7d69f69f4b   D8DeckhouseIsNotOnReleaseChannel               9
   12h   53s             firing
   205a551243d795f3   D8LogShipperAgentNotScheduledInCluster         7
   15h   63s             firing
   2e34039aa7a3018e   D8NodeIsNotUpdating                            9
   12h   47s             firing
   31baf9a70d657275   D8StrongholdClusterNotHealthy                  7
   16h   55s             firing
   ```

   For more information about alerts, see the The list of alerts section.

2. Check the Deckhouse task queue:

   ```
   d8 s queue list
   ```

   Example output (queues are empty):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

If there are many unprocessed or long-running tasks in the queue, this may indicate problems.

3. Analyze DP logs and events:
   - To view logs in real time, use the command:

     ```
     d8 k -n d8-system logs -f -l app=deckhouse
     ```

     Example output:

     ```
     {"level":"info","logger":"addon-operator","msg":"ConvergeModules task for
     OperatorStartup in phase '', trigger is Operator-
     Startup","binding":"ConvergeModules","event.type":"OperatorStartup","queue"
     :"main","task.flow":"start","task.id":"fde0eb3b-5c3e-4da6-a0d8-
     a52f8ae03428","time":"2025-11-26T08:29:33Z"}
     {"level":"warn","logger":"addon-operator.converge-
     modules","msg":"ConvergeModules: functional scheduler not
     finished","binding":"ConvergeModules","event.type":"OperatorStartup","queue
     ":"main","task.id":"fde0eb3b-5c3e-4da6-a0d8-a52f8ae03428","time":"2025-11-
     26T08:29:33Z"}
     ```

     When analyzing logs, pay special attention to warnings (`WARNING`) and error messages (`ERROR`).

   - To view events, use the command:

     ```
     d8 k -n d8-system get events
     ```

     Example output:

     ```
     LAST SEEN   TYPE      REASON              OBJECT
     MESSAGE
     11m         Warning   Unhealthy           pod/deckhouse-5886c9bd77-vgdbw
     Readiness probe failed: HTTP probe failed with statuscode: 500
     7m22s       Normal    SuccessfulDelete    replicaset/deckhouse-5886c9bd77
     Deleted pod: deckhouse-5886c9bd77-vgdbw
     7m20s       Normal    Scheduled           pod/deckhouse-6bc5c4494-fwx6z
     Successfully assigned d8-system/deckhouse-6bc5c4494-fwx6z to sandbox1-
     master-0
     7m20s       Normal    Pulling             pod/deckhouse-6bc5c4494-fwx6z
     Pulling image "dev-registry.deckhouse.io/sys/deckhouse-
     oss@sha256:17ac07634e17422df52720264cddec3916ed6985a77782dc8a24fe5352290e6e
     "
     ```

   When analyzing events, pay special attention to those with the `Warning` type.

Deckhouse

How can I check DP versions available in different release channels?

For information about the current DP versions across all release channels, visit releases.deckhouse.io.

How to check the job queue in Deckhouse?

How to check the status of all Deckhouse task queues?

To view the status of all Deckhouse job queues, run the following command:

```
d8 s queue list
```

Example output (queues are empty):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

How to view the status of the main task queue?

To view the status of the Deckhouse `main` task queue, run the following command:

```
d8 s queue main
```

Example output (38 tasks in the `main` queue):

```
Queue 'main': length 38, status: 'run first task'
```

Example output (the `main` queue is empty):

```
Queue 'main': length 0, status: 'waiting for task 0s'
```

What should I do if DP is not receiving updates from the configured channel?

- Ensure the correct release channel is configured.
- Check that DNS resolution for the Deckhouse image registry is working correctly.

  Get and compare the IP addresses of `registry.deckhouse.io` from both a node and the `deckhouse` Pod. They must match.

  Example of obtaining an IP of `registry.deckhouse.io` from a node:

  ```
  getent ahosts registry.deckhouse.io
  ```

  Example output:

  ```
  185.193.90.38    STREAM registry.deckhouse.io
  185.193.90.38    DGRAM
  185.193.90.38    RAW
  ```

  Example of obtaining an IP of `registry.deckhouse.io` from the `deckhouse` Pod:

  ```
  d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- getent ahosts
  registry.deckhouse.io
  ```

  Example output:

  ```
  185.193.90.38    STREAM registry.deckhouse.io
  185.193.90.38    DGRAM  registry.deckhouse.io
  ```

  If the resulted IPs do not match, check DNS settings on the node. Pay attention to the `search` domain list in `/etc/resolv.conf`, which affects name resolution in the `deckhouse` Pod. If the `search` parameter in `/etc/resolv.conf` file specifies a domain with wildcard DNS resolution configured, this may lead to incorrect IP address resolution for the Deckhouse image registry (see example below).

Example DNS settings that may cause issues resolving the DP image registry IP address

Below is an example of how DNS settings may result in different resolution behavior on the node and in a Kubernetes Pod:

- Example of `/etc/resolv.conf` on the node:

  ```
  nameserver 10.0.0.10
  search company.my
  ```

  > **i** On nodes, the default `ndot` setting is **1** (`options ndots:1`), while in Kubernetes Pods, it's **5**. This causes different resolution logic for DNS names with 5 or fewer dots on a node and on the Pod.

- The DNS zone `company.my` has a wildcard entry `*.company.my` that resolves to `10.0.0.100`. This means any undefined DNS name in the `company.my` zone resolves to `10.0.0.100`.

Taking into account the `search` parameter in `/etc/resolv.conf`, when accessing `registry.deckhouse.io` from a node, the system will attempt to resolve the IP address for `registry.deckhouse.io` (because it considers it fully qualified due to the default `options ndots:1` setting).

However, when accessing `registry.deckhouse.io` from a Kubernetes Pod, considering the `options ndots:5` setting used by default in Kubernetes and the `search` parameter, the system will first attempt to resolve the name `registry.deckhouse.io.company.my`. This name will resolve to the IP address `10.0.0.100` because, according to the `company.my` DNS zone's wildcard configuration, `*.company.my` is resolved to `10.0.0.100`. As a result, the Pod will fail to connect to the `registry.deckhouse.io` host and will be unable to download information about available Deckhouse updates.

Does Deckhouse support realtime (rt) and low-latency Linux kernels?

In general, realtime (rt) and lowlatency kernels are supported with no extra configuration needed for Deckhouse Platform. Such kernels have been tested with DP EE version v1.75.7 and above on the following OSes and kernels:

- CentOS 9 Stream, linux-5.14.0-706.el9.x86_64+rt
- RedOS 8, linux-6.12.85-1.red80.x86_64-rt
- Astra Linux 1.7.5, linux-5.15-lowlatency

How can I know when a new DP version is available for the cluster?

As soon as a new version appears on the configured release channel:

- The `DeckhouseReleaseIsWaitingManualApproval` alert will appear if the cluster is in manual update mode.
- A new DeckhouseRelease custom resource will be created. To see the list of releases, run `d8 k get deckhousereleases`. If the new version is in `Pending` state, it means it hasn't been installed yet. Possible reasons:
  - Manual update mode is enabled.
  - Automatic update mode is enabled and update windows are scheduled, but the window hasn't started yet.
  - Automatic update mode is enabled and update windows are not scheduled, but the update is delayed by a random period to reduce load on the container image registry. The `status.message` field of the DeckhouseRelease resource will show a corresponding message.
  - The `update.notification.minimalNotificationTime` parameter is set, and the delay period hasn't elapsed.

How can I receive information about upcoming updates in advance?

You can get information about upcoming minor DP version updates on the release channel in one of the following ways:

- Enable manual update mode. A new DeckhouseRelease resource and the `DeckhouseReleaseIsWaitingManualApproval` alert will appear when a new version is available.
- Enable automatic update mode and set a delay using the `minimalNotificationTime` parameter. A new DeckhouseRelease resource will appear when a new version is available. If you also set a webhook URL in the `update.notification.webhook` parameter, a notification will be sent about the upcoming update.

How can I tell that the update was successful?

If the `DeckhouseUpdating` alert is gone, the update has finished.

You can also check the status of DP releases in the cluster with the following command:

```
d8 k get deckhouserelease
```

Example output:

```
NAME       PHASE        TRANSITIONTIME   MESSAGE
v1.46.8    Superseded   13d
v1.46.9    Superseded   11d
v1.47.0    Superseded   4h12m
v1.47.1    Deployed     4h12m
```

The `Deployed` status means the cluster has switched to the corresponding version, but it doesn't guarantee that the update has been successful.

To ensure the update completed successfully, check the state of the `deckhouse` Pod with the following command:

```
d8 k -n d8-system get pods -l app=deckhouse
```

Example output:

```
NAME                     READY   STATUS    RESTARTS   AGE
deckhouse-7844b47bcd-qtbx9   1/1     Running   0          1d
```

- If the Pod is `Running` and shows `1/1` under `READY`, it means the update completed successfully.
- If the Pod is `Running` but shows `0/1` under `READY`, it means the update is still in progress. If it stays like this for more than 20–30 minutes, it may indicate a problem with DP that requires investigation.
- If the Pod is not `Running`, it may indicate a problem with DP that requires investigation.

If something goes wrong

- Check the logs using the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
  ```

- Collect debug information and contact DP technical support.

How can I tell that an update is in progress?

During an update:

- The `DeckhouseUpdating` alert is active.
- The `deckhouse` Pod is not in the `Ready` state. If the Pod stays in a non-`Ready` state for a long time, it may indicate an issue with DP that requires investigation.

What to do if there are problems updating DP?

Deckhouse Platform update fails, one or more Deckhouse pods are in an unworkable state

If the Deckhouse Platform update fails, one or more Deckhouse pods in the `d8-system` namespace are in an unworkable state. Perform the following steps:

1. Check the Deckhouse logs using the command:

   ```
   d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
   ```

   If there are any problems, information about them will be included in the output. When analyzing logs, pay special attention to warnings (`WARNING`) and error messages (`ERROR`).

2. Check Deckhouse events using the command:

```
d8 k -n d8-system describe po -l app=deckhouse | awk '
/^Name:/ {
    pod = $2;
    print "=== " pod " ===";
    in_events = 0
}
/Events:/ {
    in_events = 1;
    next
}
in_events && /^$/ {
    in_events = 0;
    print "---"
}
in_events && !/^Events:/ {
    print $0
}
' | sed '/^---$/N;/^\n$/D'
```

Pod events contain key information about problems (e.g., planning errors, image loading errors, etc.). When analyzing events, pay special attention to those with the `Warning` type.

Example output:

```
Type     Reason     Age                     From     Message
----     ------     ----                    ----     -------
Warning  Unhealthy  4m44s (x1918 over 154m)  kubelet  Readiness probe failed:
HTTP probe failed with statuscode: 500
```

DP update is stuck in the Release is suspended status

The status `Release is suspended` indicates that it has been postponed and is currently unavailable (not recommended) for installation. In this case, it is recommended to remain on the latest available release or on the one currently installed (it will have the status `Deployed`).

To view the list of releases, use the command:

```
d8 k get deckhousereleases.deckhouse.io
```

Example output:

```
NAME       PHASE        TRANSITIONTIME   MESSAGE
v1.69.13   Skipped      3h46m
v1.69.14   Skipped      3h46m
v1.69.15   Skipped      3h46m
v1.69.16   Superseded   160m
v1.70.12   Suspended    49d              Release is suspended
v1.70.13   Skipped      36d
v1.70.14   Skipped      34d
v1.70.15   Skipped      28d
v1.70.16   Skipped      19d
v1.70.17   Deployed     160m
v1.71.3    Suspended    14d              Release is suspended
```

IAM

What to do if there are problems applying DexProvider settings?

If you have changed the DexProvider settings in the `user-authn` module and one of the following problems occurs:

- No changes are visible (settings are not applied).

- When attempting to log in to the platform's web interface with any type of authorization, a `500 Internal Server Error` occurs without a detailed description.

Perform the following steps:

1. Check the status of the dex deployment:

   ```
   d8 k -n d8-user-authn get pod
   ```

   Example output:

   ```
   NAME                                      READY   STATUS    RESTARTS   AGE
   dex-5ddb779b7d-6pbhs                      2/2     Running   0          20h
   kubeconfig-generator-7c46977b9f-5kdmc     1/1     Running   0          20h
   ```

   If the module is functioning properly and the correct configuration is specified in DexProvider, all pods will have the status `Running`.

2. Check the logs for the problematic pod:

   ```
   d8 k -n d8-user-authn logs dex-<pod-name>
   ```

   Based on the information from the logs, correct the configuration in the DexProvider resource and wait for the dex pods to restart. Within a few minutes, the pods will restart automatically, and the platform's web interface (located at `console.<CLUSTER_NAME_TEMPLATE>`) will become available and will reflect the changes made to the DexProvider resource.

Kubernetes and Scheduling

What to do when the API server is overloaded?

The following signs may indicate problems with API server load and memory consumption:

- `kubectl (d8)` responds slowly or does not respond at all (commands are executed slowly or not at all).
- Pods are recreated in the cluster for no apparent reason.

If these signs are present, perform the following actions:

1. Check the resource consumption of API server pods. To do this, use the command:

   ```
   d8 k -n kube-system top po -l component=kube-apiserver
   ```

   Pay attention to `MEMORY` consumption and `CPU`.

   Example output:

   ```
   NAME                                CPU(cores)   MEMORY(bytes)
   kube-apiserver-sandbox1-master-0    251m         1476Mi
   ```

2. Check the metrics in Grafana.

   To view the metrics, open the dashboard "Home" → "Dashboards" → "Kubernetes Cluster" → "Control Plane Status". Review the graphs related to the API server ("Kube-apiserver CPU Usage", "Kube-apiserver Memory Usage", "Kube-apiserver latency", etc.).

3. Review the API server audit logs to identify the source of high memory consumption. One common cause of high memory consumption is a large number of requests.

How to check the Kubernetes version in use?

To check the Kubernetes version in use, run the command:

```
d8 k get nodes
```

Example output:

```
NAME                  STATUS   ROLES                  AGE    VERSION
frontend-0            Ready    frontend               118d   v1.31.9
master-0              Ready    control-plane,master   118d   v1.31.9
master-1              Ready    control-plane,master   118d   v1.31.9
master-2              Ready    control-plane,master   118d   v1.31.9
system-0              Ready    system                 118d   v1.31.9
system-1              Ready    system                 118d   v1.31.9
worker-0              Ready    worker                 37d    v1.31.9
worker-1              Ready    worker                 19d    v1.31.9
```

Network

Is the ingress-nginx module supported? Will it affect passing a PCI DSS audit?

The `ingress-nginx` module remains supported by Deckhouse Platform for the entire platform support lifecycle and does not depend on the upstream project's status. The Deckhouse team tracks vulnerabilities in the controller and related components (NGINX, Lua modules, base images), delivers fixes in DP releases, and provides a migration path toward Gateway API.

Flant is the vendor responsible for support. Together with ongoing module maintenance and vulnerability management, this allows passing a PCI DSS audit and meets expectations for vendor accountability for a platform component.

Virtualization

How to use Ansible to provision virtual machines?

Ansible is an automation tool for running tasks on remote servers over SSH. This example shows how to use Ansible with virtual machines (VMs) in the `demo-app` project.

The example assumes that:

- `demo-app` namespace contains a VM named `frontend`.
- VM has a `cloud` user with SSH access.
- Private SSH key on the machine where Ansible runs is stored in `/home/user/.ssh/id_rsa`.

1. Create an `inventory.yaml` file:

   ```
   ---
   all:
     vars:
       ansible_ssh_common_args: '-o ProxyCommand="d8 v port-forward --stdio=true %h
   %p"'
       # Default user for SSH access.
       ansible_user: cloud
       # Path to private key.
       ansible_ssh_private_key_file: /home/user/.ssh/id_rsa
     hosts:
       # Host name in the format <VM_NAME>.<NAMESPACE>.
       frontend.demo-app:
   ```

2. Check the virtual machine `uptime`:

   ```
   ansible -m shell -a "uptime" -i inventory.yaml all

   # frontend.demo-app | CHANGED | rc=0 >>
   # 12:01:20 up 2 days,  4:59,  0 users,  load average: 0.00, 0.00, 0.00
   ```

If you do not want to use an inventory file, pass all parameters on the command line:

```
ansible -m shell -a "uptime" \
  -i "frontend.demo-app," \
  -e "ansible_ssh_common_args='-o ProxyCommand=\"d8 v port-forward --stdio=true %h
%p\"'" \
  -e "ansible_user=cloud" \
  -e "ansible_ssh_private_key_file=/home/user/.ssh/id_rsa" \
  all
```

How to automatically generate inventory for Ansible?

Instead of manually creating an inventory file, you can use the `d8 v ansible-inventory` command, which automatically generates an Ansible inventory from virtual machines in the specified namespace. The command is compatible with the ansible inventory script interface.

The `d8 v ansible-inventory` command requires `d8` v0.27.0 or higher. The command works only for virtual machines that have the main cluster network (Main) connected.

Only machines in the `Running` phase that have an assigned IP address get into the inventory. Host names are formatted as `<VM_NAME>.<NAMESPACE>` (for example, `frontend.demo-app`).

1. Optionally set host variables via annotations (for example, the SSH user):

   ```
   d8 k -n demo-app annotate vm frontend vars.ansible.deckhouse.io/ansible_user="cloud"
   ```

2. Run Ansible with a dynamically generated inventory:

   ```
   ANSIBLE_INVENTORY_ENABLED=yaml ansible -m shell -a "uptime" all -i <(d8 v ansible-
   inventory -n demo-app -o yaml)
   ```

> **i** The `<(...)` construct is necessary because Ansible expects a file or script as the source of the host list. Simply specifying the command in quotes won't work, because Ansible tries to execute the string as a script. The `<(...)` construct passes the command output as a file that Ansible can read.

1. Or save the inventory to a file and run the check:

   ```
   d8 v ansible-inventory --list -o yaml -n demo-app > inventory.yaml
   ansible -m shell -a "uptime" -i inventory.yaml all
   ```

How to use cloud-init to configure virtual machines?

Cloud-init is used for initial guest OS configuration on first boot. The configuration is written in YAML and starts with the `#cloud-config` directive.

> **⚠** When using cloud images (for example, official distribution images), you must provide a cloud-init configuration. Without it, some distributions do not configure network connectivity, and the virtual machine becomes unreachable on the network, even if the main network (Main) is attached.
>
> In addition, cloud images do not allow login by default — you must either add SSH keys for the default user or create a new user with SSH access. Otherwise, you will not be able to access the virtual machine.

Updating and installing packages

Example `cloud-config` for updating the system and installing packages from a list:

```
#cloud-config
# Update package lists.
package_update: true
# Upgrade installed packages to latest versions.
package_upgrade: true
# List of packages to install.
packages:
  - nginx
  - curl
  - htop
# Commands to run after package installation.
runcmd:
  - systemctl enable --now nginx.service
```

Creating a user

Example `cloud-config` for creating a local user with a password and SSH key:

```
#cloud-config
# List of users to create.
users:
   # Username.
  - name: cloud
    # Password hash.
    passwd: "<PASSWORD_HASH>"
    # Do not lock the account.
    lock_passwd: false
    # Sudo privileges without a password prompt.
    sudo: ALL=(ALL) NOPASSWD:ALL
    # Default shell.
    shell: /bin/bash
    # SSH keys for access.
    ssh-authorized-keys:
      - <SSH_PUBLIC_KEY>
# Allow password authentication via SSH.
ssh_pwauth: true
```

To generate a password hash for the `passwd` field, run:

```
mkpasswd --method=SHA-512 --rounds=4096
```

Creating a file with required permissions

Example `cloud-config` for creating a file with specified access permissions:

```
#cloud-config
# List of files to create.
write_files:
   # File path.
  - path: /opt/scripts/start.sh
    # File content.
    content: |
      #!/bin/bash
      echo "Starting application"
    # File owner, user and group.
    owner: cloud:cloud
    # Access permissions in octal format.
    permissions: '0755'
```

Configuring disk and filesystem

Example `cloud-config` for disk partitioning, filesystem creation, and mounting:

```
#cloud-config
# Disk partitioning setup.
disk_setup:
  # Disk device.
  /dev/sdb:
    # Partition table type, gpt or mbr.
    table_type: gpt
    # Automatically create partitions.
    layout: true
    # Do not overwrite existing partitions.
    overwrite: false

# Filesystem setup.
fs_setup:
    # Filesystem label.
  - label: data
    # Filesystem type.
    filesystem: ext4
    # Partition device.
    device: /dev/sdb1
    # Automatically detect partition.
    partition: auto

# Filesystem mounting.
mounts:
  # [device, mount_point, fs_type, options, dump, pass]
  - ["/dev/sdb1", "/mnt/data", "ext4", "defaults", "0", "2"]
```

Configuring network interfaces for additional networks

> **i** The settings described in this section apply only to additional networks. The main network (Main) is configured automatically via cloud-init and does not require manual configuration.

Additional networks are configured manually via cloud-init. The `write_files` block creates the configuration files, and the `runcmd` block applies the settings.

Connecting additional networks to a virtual machine is covered in Additional network interfaces.

The following examples cover common ways to configure networking in the guest OS:

- systemd-networkd
- Netplan (Ubuntu)
- ifcfg (RHEL/CentOS)
- Alpine Linux

Example `cloud-config` for distributions that use `systemd-networkd` (Debian, CoreOS, and others):

```
#cloud-config
write_files:
  - path: /etc/systemd/network/10-eth1.network
    content: |
      [Match]
      Name=eth1

      [Network]
      Address=192.168.1.10/24
      Gateway=192.168.1.1
      DNS=8.8.8.8

runcmd:
  - systemctl restart systemd-networkd
```

Example `cloud-config` for Ubuntu and other systems that use `Netplan`:

```
#cloud-config
write_files:
  - path: /etc/netplan/99-custom.yaml
    content: |
      network:
        version: 2
        ethernets:
          eth1:
            addresses:
              - 10.0.0.5/24
            gateway4: 10.0.0.1
            nameservers:
              addresses: [8.8.8.8]
          eth2:
            dhcp4: true

runcmd:
  - netplan apply
```

Example `cloud-config` for RHEL-compatible distributions that use the `ifcfg` scheme and `NetworkManager`:

```
#cloud-config
write_files:
  - path: /etc/sysconfig/network-scripts/ifcfg-eth1
    content: |
      DEVICE=eth1
      BOOTPROTO=none
      ONBOOT=yes
      IPADDR=192.168.1.10
      PREFIX=24
      GATEWAY=192.168.1.1
      DNS1=8.8.8.8

runcmd:
  - nmcli connection reload
  - nmcli connection up eth1
```

Example `cloud-config` for distributions that use the traditional `/etc/network/interfaces` format (Alpine and similar):

```
#cloud-config
write_files:
  - path: /etc/network/interfaces
    append: true
    content: |
      auto eth1
      iface eth1 inet static
          address 192.168.1.10
          netmask 255.255.255.0
          gateway 192.168.1.1

runcmd:
  - /etc/init.d/networking restart
```

How to increase the DVCR size?

The DVCR volume size is set in the `virtualization` module ModuleConfig (`spec.settings.dvcr.storage.persistentVolumeClaim.size`). The new value must be greater than the current one.

1. Check the current DVCR size:

   ```
   d8 k get mc virtualization -o
   jsonpath='{.spec.settings.dvcr.storage.persistentVolumeClaim}'
   ```

   Example output:

   ```
   {"size":"58G","storageClass":"linstor-thick-data-r1"}
   ```

2. Increase `size` using `patch` (set the value you need):

   ```
   d8 k patch mc virtualization \
     --type merge -p '{"spec": {"settings": {"dvcr": {"storage":
   {"persistentVolumeClaim": {"size":"59G"}}}}}}'
   ```

   Example output:

   ```
   moduleconfig.deckhouse.io/virtualization patched
   ```

3. Verify that ModuleConfig shows the new size:

   ```
   d8 k get mc virtualization -o
   jsonpath='{.spec.settings.dvcr.storage.persistentVolumeClaim}'
   ```

   Example output:

   ```
   {"size":"59G","storageClass":"linstor-thick-data-r1"}
   ```

4. Check the current DVCR status:

   ```
   d8 k get pvc dvcr -n d8-virtualization
   ```

   Example output:

   ```
   NAME STATUS VOLUME                                     CAPACITY    ACCESS MODES
   STORAGECLASS           AGE
   dvcr Bound  pvc-6a6cedb8-1292-4440-b789-5cc9d15bbc6b  57617188Ki  RWO
   linstor-thick-data-r1  7d
   ```

How to change the DVCR StorageClass when a PVC already exists?

> ⚠ You can change the DVCR storage StorageClass only by recreating the PVC. All images previously loaded into DVCR are lost, along with data for existing ClusterVirtualImage and VirtualImage resources.

The `spec.settings.dvcr.storage.persistentVolumeClaim.storageClassName` field in the `virtualization` module ModuleConfig sets the StorageClass for the virtual machine image storage volume (DVCR). While a PVC for that volume exists in the `d8-virtualization` namespace, you cannot change the field via the API.

You cannot change `storageClassName` on an existing PVC in place, and DVCR data is not migrated between storage classes.

To change the DVCR StorageClass, perform the following steps:

1. Stop DVCR:

   ```
   d8 k -n d8-virtualization scale deployment dvcr --replicas=0
   ```

2. List PVCs in the `d8-virtualization` namespace and find the PVC for the DVCR volume:

```
d8 k get pvc -n d8-virtualization
```

3. Delete the PVC you found. Replace `<PVC_NAME>` with the resource name. If the command fails because of insufficient permissions, run it as `system:sudouser`:

```
d8 k --as system:sudouser -n d8-virtualization delete pvc/<PVC_NAME>
```

4. Set the new StorageClass in ModuleConfig, substituting the class you need for `<STORAGE_CLASS_NAME>`:

```
d8 k patch mc virtualization --type merge -p '{"spec":{"settings":{"dvcr":{"storage":
{"persistentVolumeClaim":{"storageClassName":"<STORAGE_CLASS_NAME>"}}}}}}'
```

   Example output:

```
moduleconfig.deckhouse.io/virtualization patched
```

5. Start DVCR:

```
d8 k -n d8-virtualization scale deployment dvcr --replicas=1
```

6. Verify the PVC:

```
d8 k get pvc -n d8-virtualization
```

   Example output:

```
NAME   STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS          VOLUMEATTRIBUTESCLASS   AGE
dvcr   Bound    pvc-b43f2e33-32cc-435a-aa1d-b53df35b030a   100Gi      RWO
linstor-thin-r1-hdd   <unset>                 34s
```

> **i** The storage for the chosen StorageClass must be reachable from the nodes where DVCR runs: system nodes, or worker nodes if the cluster has no system nodes.

How to install an operating system in a virtual machine from an ISO image?

Below is a typical Windows guest OS installation scenario from an ISO image. Before you begin, host the ISO on an HTTP endpoint reachable from the cluster.

1. Create an empty VirtualDisk for OS installation:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: win-disk
  namespace: default
spec:
  persistentVolumeClaim:
    size: 100Gi
    storageClassName: local-path
```

2. Create ClusterVirtualImage resources for the Windows OS ISO and the VirtIO driver ISO:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: ClusterVirtualImage
metadata:
  name: win-11-iso
spec:
  dataSource:
    type: HTTP
    http:
      url: "http://example.com/win11.iso"
```

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: ClusterVirtualImage
metadata:
  name: win-virtio-iso
spec:
  dataSource:
    type: HTTP
    http:
      url: "https://fedorapeople.org/groups/virt/virtio-win/direct-
downloads/stable-virtio/virtio-win.iso"
```

3. Create a virtual machine (VM):

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: win-vm
  namespace: default
  labels:
    vm: win
spec:
  virtualMachineClassName: generic
  runPolicy: Manual
  osType: Windows
  bootloader: EFI
  cpu:
    cores: 6
    coreFraction: 50%
  memory:
    size: 8Gi
  enableParavirtualization: true
  blockDeviceRefs:
    - kind: VirtualDisk
      name: win-disk
    - kind: ClusterVirtualImage
      name: win-11-iso
    - kind: ClusterVirtualImage
      name: win-virtio-iso
```

4. Start the virtual machine:

```
d8 v start win-vm
```

5. Connect to the VM console and complete the OS installation and VirtIO drivers using the graphical installer.

   VNC connection:

```
d8 v vnc -n default win-vm
```

6. After the installation is complete, restart the virtual machine.

7. For further work, connect via VNC again:

```
d8 v vnc -n default win-vm
```

How to create a golden image for Linux?

A golden image is a pre-configured virtual machine (VM) image that can be used to quickly create new VMs with pre-installed software and settings.

1. Create a virtual machine, install the required software on it, and perform all necessary configurations.
2. Install and configure qemu-guest-agent (recommended):
   - For RHEL/CentOS:

     ```
     yum install -y qemu-guest-agent
     ```

   - For Debian/Ubuntu:

     ```
     apt-get update
     apt-get install -y qemu-guest-agent
     ```

3. Enable and start the service:

   ```
   systemctl enable qemu-guest-agent
   systemctl start qemu-guest-agent
   ```

4. Set the machine run policy to `AlwaysOnUnlessStoppedManually`, otherwise you will not be able to shut it down.
5. Prepare the image. Clean unused filesystem blocks:

   ```
   fstrim -v /
   fstrim -v /boot
   ```

6. Clean network settings:
   - For RHEL:

     ```
     nmcli con delete $(nmcli -t -f NAME,DEVICE con show | grep -v ^lo: | cut -d: -f1)
     rm -f /etc/sysconfig/network-scripts/ifcfg-eth*
     ```

   - For Debian/Ubuntu:

     ```
     rm -f /etc/network/interfaces.d/*
     ```

7. Clean system identifiers:

   ```
   echo -n > /etc/machine-id
   rm -f /var/lib/dbus/machine-id
   ln -s /etc/machine-id /var/lib/dbus/machine-id
   ```

8. Remove the SSH host keys:

   ```
   rm -f /etc/ssh/ssh_host_*
   ```

9. Clean the systemd journal:

   ```
   journalctl --vacuum-size=100M --vacuum-time=7d
   ```

10. Clean package manager cache:

- For RHEL:

  ```
  yum clean all
  ```

- For Debian/Ubuntu:

  ```
  apt-get clean
  ```

11. Clean temporary files:

    ```
    rm -rf /tmp/*
    rm -rf /var/tmp/*
    ```

12. Clean logs:

    ```
    find /var/log -name "*.log" -type f -exec truncate -s 0 {} \;
    ```

13. Clean command history:

    ```
    history -c
    ```

14. On RHEL, reset and restore the SELinux contexts in one of two ways.

    Restore the contexts right away:

    ```
    restorecon -R /
    ```

    Or schedule a relabel for the next boot:

    ```
    touch /.autorelabel
    ```

15. Verify that `/etc/fstab` references UUID or `LABEL` rather than names like `/dev/sdX`:

    ```
    blkid
    cat /etc/fstab
    ```

16. Reset cloud-init state (logs and seed):

    ```
    cloud-init clean --logs --seed
    ```

17. Perform final synchronization and buffer cleanup:

    ```
    sync
    echo 3 > /proc/sys/vm/drop_caches
    ```

18. Shut down the virtual machine:

    ```
    poweroff
    ```

19. Create a VirtualImage resource that references the prepared VM’s VirtualDisk:

```
d8 k apply -f -<<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: <IMAGE_NAME>
  namespace: <NAMESPACE>
spec:
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualDisk
      name: <SOURCE_DISK_NAME>
EOF
```

Or create a ClusterVirtualImage resource so the image is available cluster-wide for all projects:

```
d8 k apply -f -<<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: ClusterVirtualImage
metadata:
  name: <IMAGE_NAME>
spec:
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualDisk
      name: <SOURCE_DISK_NAME>
      namespace: <NAMESPACE>
EOF
```

Here, `<IMAGE_NAME>` is the name of the image being created, `<NAMESPACE>` is the namespace of the prepared machine, and `<SOURCE_DISK_NAME>` is the name of its disk.

20. Create a new VirtualDisk from the resulting image:

```
d8 k apply -f -<<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: <VM_DISK_NAME>
  namespace: <NAMESPACE>
spec:
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualImage
      name: <IMAGE_NAME>
EOF
```

Here, `<VM_DISK_NAME>` is the name of the disk of the new machine.

After completing these steps, you will have a golden image that can be used to quickly create new virtual machines with pre-installed software and configurations.

How to restore the cluster if images from registry.deckhouse.io cannot be pulled after a license change?

After a license change on a cluster with `containerd v1` and removal of the outdated license, images from `registry.deckhouse.io` may stop being pulled. Nodes then retain the outdated configuration file `/etc/containerd/conf.d/dvcr.toml`, which is not removed automatically. Because of it, the `registry` module does not start, and without it DVCR does not work.

Applying a NodeGroupConfiguration (NGC) manifest removes the file on the nodes. After the `registry`

module starts, delete the manifest, since this is a one-time fix.

1. Save the manifest to a file (for example, `containerd-dvcr-remove-old-config.yaml`):

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: NodeGroupConfiguration
   metadata:
     name: containerd-dvcr-remove-old-config.sh
   spec:
     weight: 32 # Must be in range 32–90.
     nodeGroups: ["*"]
     bundles: ["*"]
     content: |
       # Copyright 2023 Flant JSC
       # Licensed under the Apache License, Version 2.0 (the "License");
       # you may not use this file except in compliance with the License.
       # You may obtain a copy of the License at
       #     http://www.apache.org/licenses/LICENSE-2.0
       # Unless required by applicable law or agreed to in writing, software
       # distributed under the License is distributed on an "AS IS" BASIS,
       # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
       # See the License for the specific language governing permissions and
       # limitations under the License.

       rm -f /etc/containerd/conf.d/dvcr.toml
   ```

2. Apply the saved manifest:

   ```
   d8 k apply -f containerd-dvcr-remove-old-config.yaml
   ```

3. Verify that the `registry` module is running:

   ```
   d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
   map_values(@base64d) | .conditions = (.conditions | from_yaml)'
   ```

   Example output when the `registry` module has started successfully:

   ```
   conditions:
   # ...
     - lastTransitionTime: "..."
       message: ""
       reason: ""
       status: "True"
       type: Ready
   ```

4. Delete the one-time NodeGroupConfiguration manifest:

   ```
   d8 k delete -f containerd-dvcr-remove-old-config.yaml
   ```

The migration procedure is covered in Migrating container runtime to containerd v2.

How to redirect traffic to a virtual machine?

A virtual machine runs in a Kubernetes cluster, so traffic reaches it the same way it reaches any other workload. Routing is handled by the standard Kubernetes Service resource, which selects targets by labels.

1. Create a service with the required settings.

   For example, consider a virtual machine with the label `vm: frontend-0`, an HTTP service exposed on ports 80 and 443, and SSH access on port 22:

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: frontend-0
  namespace: dev
  labels:
    vm: frontend-0
spec: ...
```

2. To route network traffic to the virtual machine's ports, create a service:

   This service listens on ports 80 and 443 and forwards traffic to the corresponding ports of the target virtual machine. SSH access from outside is provided on port 2211:

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-0-svc
  namespace: dev
spec:
  type: LoadBalancer
  ports:
  - name: ssh
    port: 2211
    protocol: TCP
    targetPort: 22
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  - name: https
    port: 443
    protocol: TCP
    targetPort: 443
  selector:
    vm: frontend-0
```

Why is only one connection method available for a virtual machine?

You can connect to a virtual machine (VM) via the serial console (`d8 v console`) or VNC (`d8 v vnc`). These methods use different communication channels with the guest OS and depend on its configuration. Both methods are covered in Connecting to a virtual machine.

The sections below describe common situations where only one connection method works.

Why does VNC not work when the serial console is available?

VNC displays the guest OS screen and requires virtual terminal support in the kernel. The serial console works independently of the graphics subsystem.

Check in the guest OS whether virtual terminal support is enabled in the kernel configuration:

```
cat /boot/config-$(uname -r) | grep CONFIG_VT
```

The output should show `CONFIG_VT=y`:

```
CONFIG_VT=y
```

If the output shows `CONFIG_VT is not set`, rebuild the kernel with the option enabled or use an OS image with a suitable kernel configuration.

Why does the serial console not work when VNC is available?

The serial console connects to the `ttyS0` port in the guest OS. If the `getty` service for this port is not running, `d8 v console` will not show a login prompt even though VNC continues to work.

In the guest OS, enable and start the `serial-getty` service for `ttyS0`:

```
sudo systemctl enable --now serial-getty@ttyS0.service
```

Then connect to the serial console again.

How to provide a Windows answer file (Sysprep)?

Unattended Windows installation uses an answer file (`unattend.xml` or `autounattend.xml`).

The example answer file below:

- Sets the English UI language and keyboard layout.
- Connects the `VirtIO` drivers for the setup stage (the order of devices in `blockDeviceRefs` on the VirtualMachine resource must match the paths in the file).
- Creates disk layout for installation with EFI.
- Creates the `cloud` administrator and the regular `user` account.

▼

```
<p><b>Example of the contents of the autounattend.xml file…</b>&lt;/summary&gt;</p>

<div class="language-xml highlighter-rouge"><div class="highlight"><pre class="highlight"><code><span
class="cp">&lt;?xml version="1.0" encoding="utf-8"?&gt;</span> <span class="nt">&lt;unattend</span> <span
class="na">xmlns=</span><span class="s">"urn:schemas-microsoft-com:unattend"</span> <span
class="na">xmlns:wcm=</span><span class="s">"http://schemas.microsoft.com/WMIConfig/2002/State"</span>
<span class="nt">&gt;</span>   <span class="nt">&lt;settings</span> <span class="na">pass=</span><span
class="s">"offlineServicing"</span><span class="nt">&gt;&lt;/settings&gt;</span>   <span
class="nt">&lt;settings</span> <span class="na">pass=</span><span class="s">"windowsPE"</span><span
class="nt">&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
International-Core-WinPE"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"
</span> <span class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span
class="na">language=</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span
class="s">"nonSxS"</span><span class="nt">&gt;</span>
  <span class="nt">&lt;SetupUILanguage&gt;</span>
    <span class="nt">&lt;UILanguage&gt;</span>en-US<span class="nt">&lt;/UILanguage&gt;</span>
  <span class="nt">&lt;/SetupUILanguage&gt;</span>
  <span class="nt">&lt;InputLocale&gt;</span>0409:00000409<span class="nt">&lt;/InputLocale&gt;</span>
  <span class="nt">&lt;SystemLocale&gt;</span>en-US<span class="nt">&lt;/SystemLocale&gt;</span>
  <span class="nt">&lt;UILanguage&gt;</span>en-US<span class="nt">&lt;/UILanguage&gt;</span>
  <span class="nt">&lt;UserLocale&gt;</span>en-US<span class="nt">&lt;/UserLocale&gt;</span>
<span class="nt">&lt;/component&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
PnpCustomizationsWinPE"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"
</span> <span class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span
class="na">language=</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span
class="s">"nonSxS"</span><span class="nt">&gt;</span>
  <span class="nt">&lt;DriverPaths&gt;</span>
    <span class="nt">&lt;PathAndCredentials</span> <span class="na">wcm:keyValue=</span><span
class="s">"4b29ba63"</span> <span class="na">wcm:action=</span><span class="s">"add"</span><span
class="nt">&gt;</span>
      <span class="nt">&lt;Path&gt;</span>E:\amd64\w11<span class="nt">&lt;/Path&gt;</span>
    <span class="nt">&lt;/PathAndCredentials&gt;</span>
    <span class="nt">&lt;PathAndCredentials</span> <span class="na">wcm:keyValue=</span><span
class="s">"25fe51ea"</span> <span class="na">wcm:action=</span><span class="s">"add"</span><span
class="nt">&gt;</span>
      <span class="nt">&lt;Path&gt;</span>E:\NetKVM\w11\amd64<span class="nt">&lt;/Path&gt;</span>
    <span class="nt">&lt;/PathAndCredentials&gt;</span>
  <span class="nt">&lt;/DriverPaths&gt;</span>
<span class="nt">&lt;/component&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
Setup"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"</span> <span
class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span class="na">language=
</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span class="s">"nonSxS"
</span><span class="nt">&gt;</span>
  <span class="nt">&lt;DiskConfiguration&gt;</span>
    <span class="nt">&lt;Disk</span> <span class="na">wcm:action=</span><span class="s">"add"</span><span
class="nt">&gt;</span>
      <span class="nt">&lt;DiskID&gt;</span>0<span class="nt">&lt;/DiskID&gt;</span>
      <span class="nt">&lt;WillWipeDisk&gt;</span>true<span class="nt">&lt;/WillWipeDisk&gt;</span>
      <span class="nt">&lt;CreatePartitions&gt;</span>
        <span class="c">&lt;!-- Recovery partition --&gt;</span>
        <span class="nt">&lt;CreatePartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
```

```
          <span class="nt">&lt;Order&gt;</span>1<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;Type&gt;</span>Primary<span class="nt">&lt;/Type&gt;</span>
          <span class="nt">&lt;Size&gt;</span>250<span class="nt">&lt;/Size&gt;</span>
        <span class="nt">&lt;/CreatePartition&gt;</span>
        <span class="c">&lt;!-- EFI system partition (ESP) --&gt;</span>
        <span class="nt">&lt;CreatePartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>2<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;Type&gt;</span>EFI<span class="nt">&lt;/Type&gt;</span>
          <span class="nt">&lt;Size&gt;</span>100<span class="nt">&lt;/Size&gt;</span>
        <span class="nt">&lt;/CreatePartition&gt;</span>
        <span class="c">&lt;!-- Microsoft reserved partition (MSR) --&gt;</span>
        <span class="nt">&lt;CreatePartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>3<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;Type&gt;</span>MSR<span class="nt">&lt;/Type&gt;</span>
          <span class="nt">&lt;Size&gt;</span>128<span class="nt">&lt;/Size&gt;</span>
        <span class="nt">&lt;/CreatePartition&gt;</span>
        <span class="c">&lt;!-- Windows partition --&gt;</span>
        <span class="nt">&lt;CreatePartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>4<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;Type&gt;</span>Primary<span class="nt">&lt;/Type&gt;</span>
          <span class="nt">&lt;Extend&gt;</span>true<span class="nt">&lt;/Extend&gt;</span>
        <span class="nt">&lt;/CreatePartition&gt;</span>
      <span class="nt">&lt;/CreatePartitions&gt;</span>
      <span class="nt">&lt;ModifyPartitions&gt;</span>
        <span class="c">&lt;!-- Recovery partition --&gt;</span>
        <span class="nt">&lt;ModifyPartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>1<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;PartitionID&gt;</span>1<span class="nt">&lt;/PartitionID&gt;</span>
          <span class="nt">&lt;Label&gt;</span>Recovery<span class="nt">&lt;/Label&gt;</span>
          <span class="nt">&lt;Format&gt;</span>NTFS<span class="nt">&lt;/Format&gt;</span>
          <span class="nt">&lt;TypeID&gt;</span>de94bba4-06d1-4d40-a16a-bfd50179d6ac<span
class="nt">&lt;/TypeID&gt;</span>
        <span class="nt">&lt;/ModifyPartition&gt;</span>
        <span class="c">&lt;!-- EFI system partition (ESP) --&gt;</span>
        <span class="nt">&lt;ModifyPartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>2<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;PartitionID&gt;</span>2<span class="nt">&lt;/PartitionID&gt;</span>
          <span class="nt">&lt;Label&gt;</span>System<span class="nt">&lt;/Label&gt;</span>
          <span class="nt">&lt;Format&gt;</span>FAT32<span class="nt">&lt;/Format&gt;</span>
        <span class="nt">&lt;/ModifyPartition&gt;</span>
        <span class="c">&lt;!-- MSR partition does not need to be modified --&gt;</span>
        <span class="c">&lt;!-- Windows partition --&gt;</span>
        <span class="nt">&lt;ModifyPartition</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
          <span class="nt">&lt;Order&gt;</span>3<span class="nt">&lt;/Order&gt;</span>
          <span class="nt">&lt;PartitionID&gt;</span>4<span class="nt">&lt;/PartitionID&gt;</span>
          <span class="nt">&lt;Label&gt;</span>Windows<span class="nt">&lt;/Label&gt;</span>
          <span class="nt">&lt;Letter&gt;</span>C<span class="nt">&lt;/Letter&gt;</span>
          <span class="nt">&lt;Format&gt;</span>NTFS<span class="nt">&lt;/Format&gt;</span>
        <span class="nt">&lt;/ModifyPartition&gt;</span>
      <span class="nt">&lt;/ModifyPartitions&gt;</span>
    <span class="nt">&lt;/Disk&gt;</span>
    <span class="nt">&lt;WillShowUI&gt;</span>OnError<span class="nt">&lt;/WillShowUI&gt;</span>
  <span class="nt">&lt;/DiskConfiguration&gt;</span>
  <span class="nt">&lt;ImageInstall&gt;</span>
    <span class="nt">&lt;OSImage&gt;</span>
      <span class="nt">&lt;InstallTo&gt;</span>
        <span class="nt">&lt;DiskID&gt;</span>0<span class="nt">&lt;/DiskID&gt;</span>
        <span class="nt">&lt;PartitionID&gt;</span>4<span class="nt">&lt;/PartitionID&gt;</span>
      <span class="nt">&lt;/InstallTo&gt;</span>
    <span class="nt">&lt;/OSImage&gt;</span>
  <span class="nt">&lt;/ImageInstall&gt;</span>
  <span class="nt">&lt;UserData&gt;</span>
    <span class="nt">&lt;ProductKey&gt;</span>
      <span class="nt">&lt;Key&gt;&lt;PRODUCT_KEY&gt;&lt;/Key&gt;</span>
      <span class="nt">&lt;WillShowUI&gt;</span>OnError<span class="nt">&lt;/WillShowUI&gt;</span>
    <span class="nt">&lt;/ProductKey&gt;</span>
    <span class="nt">&lt;AcceptEula&gt;</span>true<span class="nt">&lt;/AcceptEula&gt;</span>
  <span class="nt">&lt;/UserData&gt;</span>
  <span class="nt">&lt;UseConfigurationSet&gt;</span>false<span class="nt">&lt;/UseConfigurationSet&gt;
</span>
<span class="nt">&lt;/component&gt;</span>   <span class="nt">&lt;/settings&gt;</span>   <span
class="nt">&lt;settings</span> <span class="na">pass=</span><span class="s">"generalize"</span><span
class="nt">&gt;&lt;/settings&gt;</span>   <span class="nt">&lt;settings</span> <span class="na">pass=
</span><span class="s">"specialize"</span><span class="nt">&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
```

```
Deployment"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"</span> <span
class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span class="na">language=
</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span class="s">"nonSxS"
</span><span class="nt">&gt;</span>
  <span class="nt">&lt;RunSynchronous&gt;</span>
    <span class="nt">&lt;RunSynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>1<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;Path&gt;</span>powershell.exe -NoProfile -Command "$xml = [xml]::new();
$xml.Load('C:\Windows\Panther\unattend.xml'); $sb = [scriptblock]::Create(
$xml.unattend.Extensions.ExtractScript ); Invoke-Command -ScriptBlock $sb -ArgumentList $xml;"<span
class="nt">&lt;/Path&gt;</span>
    <span class="nt">&lt;/RunSynchronousCommand&gt;</span>
    <span class="nt">&lt;RunSynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>2<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;Path&gt;</span>powershell.exe -NoProfile -Command "Get-Content -LiteralPath
'C:\Windows\Setup\Scripts\Specialize.ps1' -Raw | Invoke-Expression;"<span class="nt">&lt;/Path&gt;</span>
    <span class="nt">&lt;/RunSynchronousCommand&gt;</span>
    <span class="nt">&lt;RunSynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>3<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;Path&gt;</span>reg.exe load "HKU\DefaultUser" "C:\Users\Default\NTUSER.DAT"
<span class="nt">&lt;/Path&gt;</span>
    <span class="nt">&lt;/RunSynchronousCommand&gt;</span>
    <span class="nt">&lt;RunSynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>4<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;Path&gt;</span>powershell.exe -NoProfile -Command "Get-Content -LiteralPath
'C:\Windows\Setup\Scripts\DefaultUser.ps1' -Raw | Invoke-Expression;"<span class="nt">&lt;/Path&gt;
</span>
    <span class="nt">&lt;/RunSynchronousCommand&gt;</span>
    <span class="nt">&lt;RunSynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>5<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;Path&gt;</span>reg.exe unload "HKU\DefaultUser"<span class="nt">&lt;/Path&gt;
</span>
    <span class="nt">&lt;/RunSynchronousCommand&gt;</span>
  <span class="nt">&lt;/RunSynchronous&gt;</span>
<span class="nt">&lt;/component&gt;</span>   <span class="nt">&lt;/settings&gt;</span>   <span
class="nt">&lt;settings</span> <span class="na">pass=</span><span class="s">"auditSystem"</span><span
class="nt">&gt;&lt;/settings&gt;</span>   <span class="nt">&lt;settings</span> <span class="na">pass=
</span><span class="s">"auditUser"</span><span class="nt">&gt;&lt;/settings&gt;</span>   <span
class="nt">&lt;settings</span> <span class="na">pass=</span><span class="s">"oobeSystem"</span><span
class="nt">&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
International-Core"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"</span>
<span class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span
class="na">language=</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span
class="s">"nonSxS"</span><span class="nt">&gt;</span>
  <span class="nt">&lt;InputLocale&gt;</span>0409:00000409<span class="nt">&lt;/InputLocale&gt;</span>
  <span class="nt">&lt;SystemLocale&gt;</span>en-US<span class="nt">&lt;/SystemLocale&gt;</span>
  <span class="nt">&lt;UILanguage&gt;</span>en-US<span class="nt">&lt;/UILanguage&gt;</span>
  <span class="nt">&lt;UserLocale&gt;</span>en-US<span class="nt">&lt;/UserLocale&gt;</span>
<span class="nt">&lt;/component&gt;</span>
<span class="nt">&lt;component</span> <span class="na">name=</span><span class="s">"Microsoft-Windows-
Shell-Setup"</span> <span class="na">processorArchitecture=</span><span class="s">"amd64"</span> <span
class="na">publicKeyToken=</span><span class="s">"31bf3856ad364e35"</span> <span class="na">language=
</span><span class="s">"neutral"</span> <span class="na">versionScope=</span><span class="s">"nonSxS"
</span><span class="nt">&gt;</span>
  <span class="nt">&lt;UserAccounts&gt;</span>
    <span class="nt">&lt;LocalAccounts&gt;</span>
      <span class="nt">&lt;LocalAccount</span> <span class="na">wcm:action=</span><span class="s">"add"
</span><span class="nt">&gt;</span>
        <span class="nt">&lt;Name&gt;</span>cloud<span class="nt">&lt;/Name&gt;</span>
        <span class="nt">&lt;DisplayName&gt;</span>cloud<span class="nt">&lt;/DisplayName&gt;</span>
        <span class="nt">&lt;Group&gt;</span>Administrators<span class="nt">&lt;/Group&gt;</span>
        <span class="nt">&lt;Password&gt;</span>
          <span class="nt">&lt;Value&gt;&lt;ADMIN_PASSWORD&gt;&lt;/Value&gt;</span>
          <span class="nt">&lt;PlainText&gt;</span>true<span class="nt">&lt;/PlainText&gt;</span>
        <span class="nt">&lt;/Password&gt;</span>
      <span class="nt">&lt;/LocalAccount&gt;</span>
      <span class="nt">&lt;LocalAccount</span> <span class="na">wcm:action=</span><span class="s">"add"
</span><span class="nt">&gt;</span>
        <span class="nt">&lt;Name&gt;</span>User<span class="nt">&lt;/Name&gt;</span>
        <span class="nt">&lt;DisplayName&gt;</span>user<span class="nt">&lt;/DisplayName&gt;</span>
        <span class="nt">&lt;Group&gt;</span>Users<span class="nt">&lt;/Group&gt;</span>
        <span class="nt">&lt;Password&gt;</span>
          <span class="nt">&lt;Value&gt;&lt;USER_PASSWORD&gt;&lt;/Value&gt;</span>
          <span class="nt">&lt;PlainText&gt;</span>true<span class="nt">&lt;/PlainText&gt;</span>
        <span class="nt">&lt;/Password&gt;</span>
```

```
      <span class="nt">&lt;/LocalAccount&gt;</span>
    <span class="nt">&lt;/LocalAccounts&gt;</span>
  <span class="nt">&lt;/UserAccounts&gt;</span>
  <span class="nt">&lt;AutoLogon&gt;</span>
    <span class="nt">&lt;Username&gt;</span>cloud<span class="nt">&lt;/Username&gt;</span>
    <span class="nt">&lt;Enabled&gt;</span>true<span class="nt">&lt;/Enabled&gt;</span>
    <span class="nt">&lt;LogonCount&gt;</span>1<span class="nt">&lt;/LogonCount&gt;</span>
    <span class="nt">&lt;Password&gt;</span>
      <span class="nt">&lt;Value&gt;&lt;ADMIN_PASSWORD&gt;&lt;/Value&gt;</span>
      <span class="nt">&lt;PlainText&gt;</span>true<span class="nt">&lt;/PlainText&gt;</span>
    <span class="nt">&lt;/Password&gt;</span>
  <span class="nt">&lt;/AutoLogon&gt;</span>
  <span class="nt">&lt;OOBE&gt;</span>
    <span class="nt">&lt;ProtectYourPC&gt;</span>3<span class="nt">&lt;/ProtectYourPC&gt;</span>
    <span class="nt">&lt;HideEULAPage&gt;</span>true<span class="nt">&lt;/HideEULAPage&gt;</span>
    <span class="nt">&lt;HideWirelessSetupInOOBE&gt;</span>true<span
class="nt">&lt;/HideWirelessSetupInOOBE&gt;</span>
    <span class="nt">&lt;HideOnlineAccountScreens&gt;</span>false<span
class="nt">&lt;/HideOnlineAccountScreens&gt;</span>
  <span class="nt">&lt;/OOBE&gt;</span>
  <span class="nt">&lt;FirstLogonCommands&gt;</span>
    <span class="nt">&lt;SynchronousCommand</span> <span class="na">wcm:action=</span><span
class="s">"add"</span><span class="nt">&gt;</span>
      <span class="nt">&lt;Order&gt;</span>1<span class="nt">&lt;/Order&gt;</span>
      <span class="nt">&lt;CommandLine&gt;</span>powershell.exe -NoProfile -Command "Get-Content -
LiteralPath 'C:\Windows\Setup\Scripts\FirstLogon.ps1' -Raw | Invoke-Expression;"<span
class="nt">&lt;/CommandLine&gt;</span>
    <span class="nt">&lt;/SynchronousCommand&gt;</span>
  <span class="nt">&lt;/FirstLogonCommands&gt;</span>
<span class="nt">&lt;/component&gt;</span>   <span class="nt">&lt;/settings&gt;</span> <span
class="nt">&lt;/unattend&gt;</span> </code></pre></div>    </div>

<p>&lt;/details&gt;</p>

<p>Replace <code class="language-plaintext highlighter-rouge">&lt;PRODUCT_KEY&gt;</code> with your
Windows product key, and <code class="language-plaintext highlighter-rouge">&lt;ADMIN_PASSWORD&gt;</code>
and <code class="language-plaintext highlighter-rouge">&lt;USER_PASSWORD&gt;</code> with the passwords of
the accounts being created. Windows reads these passwords from the file in plain text, so don’t leave the
finished answer file on a shared resource.</p>

<ol>
  <li>
    <p>Save the answer file as <code class="language-plaintext highlighter-rouge">autounattend.xml</code>
(use the example above or adjust it to your needs).</p>
  </li>
  <li>
    <p>Create a secret with the type <code class="language-plaintext highlighter-
rouge">provisioning.virtualization.deckhouse.io/sysprep</code>:</p>

    <div class="language-shell highlighter-rouge"><div class="highlight"><pre class="highlight"><code>d8
k create secret generic sysprep-config <span class="nt">--type</span><span class="o">=</span><span
class="s2">"provisioning.virtualization.deckhouse.io/sysprep"</span> <span class="nt">--from-file</span>
<span class="o">=</span>./autounattend.xml </code></pre></div>        </div>
  </li>
  <li>
    <p>Create a virtual machine that will use the answer file during installation. Specify <code
class="language-plaintext highlighter-rouge">provisioning</code> with type <code class="language-
plaintext highlighter-rouge">SysprepRef</code> in the specification. If necessary, add other Base64-
encoded files to the specification required for the answer file scripts to run successfully.</p>

    <div class="language-yaml highlighter-rouge"><div class="highlight"><pre class="highlight"><code>
<span class="na">apiVersion</span><span class="pi">:</span> <span
class="s">virtualization.deckhouse.io/v1alpha2</span> <span class="na">kind</span><span class="pi">:
</span> <span class="s">VirtualMachine</span> <span class="na">metadata</span><span class="pi">:</span>
<span class="na">name</span><span class="pi">:</span> <span class="s">win-vm</span>   <span
class="na">namespace</span><span class="pi">:</span> <span class="s">default</span>   <span
class="na">labels</span><span class="pi">:</span>
<span class="na">vm</span><span class="pi">:</span> <span class="s">win</span> <span
class="na">spec</span><span class="pi">:</span>   <span class="na">virtualMachineClassName</span><span
class="pi">:</span> <span class="s">generic</span>   <span class="na">provisioning</span><span
class="pi">:</span>
<span class="na">type</span><span class="pi">:</span> <span class="s">SysprepRef</span>
<span class="na">sysprepRef</span><span class="pi">:</span>
  <span class="na">kind</span><span class="pi">:</span> <span class="s">Secret</span>
  <span class="na">name</span><span class="pi">:</span> <span class="s">sysprep-config</span>   <span
class="na">runPolicy</span><span class="pi">:</span> <span class="s">AlwaysOn</span>   <span
class="na">osType</span><span class="pi">:</span> <span class="s">Windows</span>   <span
class="na">bootloader</span><span class="pi">:</span> <span class="s">EFI</span>   <span
class="na">cpu</span><span class="pi">:</span>
<span class="na">cores</span><span class="pi">:</span> <span class="m">6</span>
<span class="na">coreFraction</span><span class="pi">:</span> <span class="s">50%</span>   <span
```

```
class="na">memory</span><span class="pi">:</span>
<span class="na">size</span><span class="pi">:</span> <span class="s">8Gi</span>   <span
class="na">enableParavirtualization</span><span class="pi">:</span> <span class="kc">true</span>   <span
class="na">blockDeviceRefs</span><span class="pi">:</span>
<span class="pi">-</span> <span class="na">kind</span><span class="pi">:</span> <span
class="s">VirtualDisk</span>
  <span class="na">name</span><span class="pi">:</span> <span class="s">win-disk</span>
<span class="pi">-</span> <span class="na">kind</span><span class="pi">:</span> <span
class="s">ClusterVirtualImage</span>
  <span class="na">name</span><span class="pi">:</span> <span class="s">win-11-iso</span>
<span class="pi">-</span> <span class="na">kind</span><span class="pi">:</span> <span
class="s">ClusterVirtualImage</span>
  <span class="na">name</span><span class="pi">:</span> <span class="s">win-virtio-iso</span> </code>
</pre></div>        </div>
  </li>
</ol>
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Release notes for Deckhouse Platform

> Patch release descriptions are generated automatically. To expand a description, click the version number.

Version 1.78

Important

- The v1 implementation of the container registry management, used up to and including DP 1.77 (the `Proxy` and `Local` modes set in the `registry` section of the `deckhouse` ModuleConfig) has been removed. A cluster can no longer be installed in these modes, and the update to DP 1.78 is blocked while the cluster runs `Proxy` or `Local`, or `Direct` without the `registry` ModuleConfig configured. Migrate the cluster to the `registry` module on DP 1.77 before updating; the procedures are described in the module FAQ. Update to DP 1.78 only from DP 1.77.1 or later: that is the first release that checks which registry implementation the cluster runs. An update that skips it, for example an LTS release hop or an update constraint from an earlier release, can install DP 1.78 on a cluster still in `Proxy` or `Local` mode, which is not supported.
- In Deckhouse Commander, checking, converging and deleting a cluster whose stored resources still set `registry.mode` to `Proxy` or `Local` in the `deckhouse` ModuleConfig now stop with an error and change nothing. Before these operations, bring the resources in line with the cluster: replace the `deckhouse` ModuleConfig in them with the cluster's own and add the cluster's `registry` ModuleConfig, if it has one.

Version 1.77

Important

- Support for Kubernetes 1.36 has been added, while support for Kubernetes 1.31 has been discontinued. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.34.
- Control plane components will be restarted during the update.
- All Ingress NGINX Controller pods will be restarted.
- If the `enableMultiTenancy` setting of the `user-authz` module is enabled, existing RoleBinding and ClusterRoleBinding objects that were previously ignored because of ClusterAuthorizationRule limitations become effective after the update. Before updating, review the RoleBinding and ClusterRoleBinding objects assigned to users, and modify or remove those that should not grant access.
- If the DP cluster uses custom roles based on the legacy RBACv2 scheme, the `D8UserAuthzLegacyRBACv2CustomRoleFound` alert will fire after the update. Such roles must be

migrated to the new scheme before updating to DP 1.78, otherwise the update will be blocked. The migration procedure is described in the `user-authz` module FAQ.

- If the cluster uses the v1 implementation of the container registry management (the `registry` section of the `deckhouse` ModuleConfig) in the `Proxy` or `Local` mode, or in the `Direct` mode without the `registry` ModuleConfig configured, it must be migrated to the `registry` module before updating to DP 1.78, otherwise the update will be blocked. The migration procedures are described in the `registry` module FAQ.
- If the `metallb` module configuration is managed through GitOps, temporarily disable synchronization from your repository before the update. After the update, replace the BGP configuration in your repository from ModuleConfig with the MetalLoadBalancerConfiguration, MetalLoadBalancerBGPPeer, and MetalLoadBalancerPool custom resources, and then re-enable synchronization.
- If the cluster uses NetworkPolicy objects with the `ipBlock: 0.0.0.0/0` rule, after the update they will also apply to intra-cluster traffic. It is recommended that you review such policies before the update and adjust them if necessary.
- In the AWS, Azure, and GCP providers, the default CNI has been changed to Cilium. Before creating or updating a cluster, make sure that the Linux kernel version is 5.8 or later and that network policies and firewalls do not block the UDP ports required for Cilium to operate in VXLAN mode. Failure to meet these requirements may result in cluster creation or update failures, or in loss of network connectivity between nodes. For detailed requirements, refer to “Network interaction”.
- The vSphere, Huawei Cloud, OpenStack, and VCD providers have been migrated from Terraform to OpenTofu. The migration is performed automatically and requires no manual actions.
- The Yandex Cloud and OpenStack providers have been migrated from Machine Controller Manager (MCM) to Cluster API (CAPI). The management engine is chosen per CloudEphemeral node group. Groups that already exist keep running on MCM, and new groups are created on CAPI by default, so a single cluster can contain groups of both kinds. To move an existing group, refer to “How to migrate node groups to Cluster API (CAPI)”.
- The following modules are now loaded from external sources and are developed on an independent release cycle (no manual migration actions are required):
  - Monitoring modules:
    - `extended-monitoring`
    - `log-shipper`
    - `loki`
    - `monitoring-custom`
    - `monitoring-deckhouse`
    - `monitoring-kubernetes`
    - `monitoring-ping`
    - `okmeter`
    - `operator-prometheus`
    - `prometheus`
    - `prometheus-metrics-adapter`
    - `prometheus-pushgateway`
    - `upmeter`
  - The `ingress-nginx` module.
- The `monitoring-kubernetes-control-plane` module has been merged into the `control-plane-manager` module. Monitoring of control plane components is now performed by the `control-plane-manager` module.

Deckhouse subsystem

- A policy has been added to the `deckhouse` module that prevents non-system users from setting or modifying the `heritage` label. This prevents accidental creation of resources that DP may recognize as its own.
- The reliability of updating clusters that use external modules (ModuleSource) has been improved. The update starts only after the modules have been preloaded, preventing their temporary unavailability during the update.
- Support has been added to the `registry` module for installing a cluster with the container registry operating in `Local` mode. This makes it possible to deploy static clusters without access to an external registry, including in air-gapped environments.

Kubernetes & Scheduling subsystem

- Changes in the `descheduler` module:
  - Added the `descheduler_pod_evictions_total` metric with information about the workload to which the evicted pod belongs.
  - Added the “Workload Happiness” Grafana dashboard, which displays the workload happiness score, workload distribution across nodes, and the module activity history (evictions, strategy execution, and timeline).
- Changes in the `vertical-pod-autoscaler` module:
  - Added the `VerticalPodAutoscalerDeprecatedUpdateModeAuto` alert and the “VPA Deprecated Update Mode” Grafana dashboard to detect VerticalPodAutoscaler objects that use the deprecated `updateMode: Auto mode`.
  - Improved the reliability of the `vertical-pod-autoscaler` webhook deployment. Fixed a component startup ordering issue that could cause errors during updates.
- Changes in the `control-plane-manager` module:
  - The module has been migrated to a new internal implementation for managing control plane components. Added the new ControlPlaneNode and ControlPlaneOperation custom resources to track node state and control plane component update operations.
  - Added automatic etcd defragmentation, which is enabled by default for multi-master clusters and runs when fragmentation exceeds 40%. A defragmentation schedule can be configured in the module settings.
  - Several settings have been moved to the module ModuleConfig. Automatic migration and alerts notifying about the need to update the configuration have also been added:
    - Kubernetes API publishing settings (`publishAPI`) from the `user-authn` module.
    - The `encryptionAlgorithm` parameter from ClusterConfiguration.
    - The `resourcesRequests` parameter for control plane components from the `global` ModuleConfig.
- Added a patch to `kube-scheduler` that prevents scheduling new pods onto nodes that are undergoing graceful shutdown.
- Accelerated etcd updates in single-master clusters. Eliminated control plane component downtime that occurred during updates.
- Reduced the risk of `kube-controller-manager` throttling under high controller load.
- Accelerated pod startup by enabling parallel pulling of multiple container images.
- Enabled the `KubeletEnsureSecretPulledImages` feature gate by default in the kubelet configuration to comply with CIS Benchmark requirements.

IAM subsystem

- Changes in the `user-authn` module:
  - Authentication protection in Dex has been strengthened: rate limiting has been added for authentication endpoints, LDAP/Crowd accounts are now automatically blocked after a series of failed sign-in attempts, and external accounts can now be blocked and unblocked manually.
  - The UserOperation resource now supports permanent user locking (`spec.lock.for: permanent`). Audit logs for account operations have also been extended: they now include information about the operation initiator, operation type, and target user.
  - Added the DexProviderCheck resource for verifying authentication providers (LDAP, OIDC, and others). It allows connection and credential issues to be diagnosed before they affect user sign-in.
  - Added the `Custom` complexity level to the password policy settings, allowing password requirements to be configured flexibly according to corporate standards.
  - Added support for specifying custom labels and annotations in the new `spec.podMetadata` field of the DexAuthenticator custom resource, for example, annotations for managing the Istio sidecar container.
  - DexAuthenticator now works correctly in namespaces with ResourceQuota and policies that require resource limits for all containers. Fixed an issue that could prevent the authenticator

pod from starting.

- Changes in the `user-authz` module:
  - The RBACv2 role model aggregation label scheme changes in DP 1.78. Added the `D8UserAuthzLegacyRBACv2CustomRoleFound` alert, which detects custom roles based on the legacy scheme. Such roles must be migrated to the new scheme before updating to DP 1.78; otherwise, they will stop working and the update will be blocked. The migration procedure is described in the module FAQ.
  - If the `enableMultiTenancy` setting is enabled, user permissions are now determined by the combination of ClusterAuthorizationRule, AuthorizationRule, and standard RBAC rules (RoleBinding/ClusterRoleBinding). After the update, existing RBAC bindings that were previously ignored because of ClusterAuthorizationRule limitations may become effective. It is recommended to review such bindings before the update. For details, refer to “Using ClusterAuthorizationRule and AuthorizationRule together with RBAC”.
  - Added the `D8UserAuthzPermissionBrowserUnavailable` alert, which notifies about the unavailability of the user permissions browsing service (`permission-browser-apiserver`).

Security subsystem

- Changes in the `admission-policy-engine` module:
  - SecurityPolicy exceptions (SecurityPolicyException) can now be configured at the individual container level instead of only for the entire pod. This allows security requirements to be relaxed for a specific sidecar or init container without removing restrictions from the entire pod. For details, see the module documentation.
  - Added the built-in ValidatingAdmissionPolicy `deny-deckhouse-finalizers.deckhouse.io`, which prevents manual removal of `deckhouse.io` finalizers from resources. This prevents incorrect object deletion and the creation of orphan resources.
- Added a mechanism to the `multitenancy-manager` module for granting cluster-wide resources to projects. DP administrators can now define which cluster resources are available to each project and specify default values for them. The mechanism can be used by any DP module to publish its own cluster resources.
- Added the ability to list resources across all accessible namespaces without cluster-wide permissions. The `d8 k get <resource> -A --scope=<accessible|projects|system|project:NAME>` command returns only the objects the user has access to instead of returning a `403 Forbidden` error. For details, see “Resource listing with access-based filtering”.
- ACL filtering of the namespace list now supports `watch` requests. The `d8 k get ns -w` command displays changes in real time without requiring the command to be re-run.
- The cainjector component of the `cert-manager` module is now enabled automatically when resources using `cert-manager` CA injection are detected. It no longer needs to be enabled manually or kept running at all times.

Cluster & Infrastructure subsystem

- Changes in `dhctl` (CLI installer):
  - Added a preflight check for disk name length in cloud providers. If the name exceeds the allowed limit, the installation is aborted before infrastructure provisioning begins.
  - Improved the interactive mode of `dhctl`: added a dedicated area for logs of current operations, and made the output more compact by displaying progress instead of a stream of service messages. Full logs are still available in debug mode and during non-interactive execution.
  - Improved the preflight check for the `deckhouse` user. Repeated bootstrap now correctly handles an existing user, and additional checks help prevent permission-related issues.
  - Fixed issues with cleaning up temporary processes and SSH connections after running the `converge` and `destroy` commands.
  - Fixed the `dhctl lock release` command in interactive mode.
  - Fixed issues in the `dhctl converge-migration` command that could cause migration failures in certain scenarios.
  - Improved the reliability of control plane `converge` operations. Checks are now performed in

a more appropriate order, and etcd status messages have become more informative.

- Changes in the `node-manager` module:
  - Added Instance API v1alpha2. Instance status and error information is now consolidated into a single resource, simplifying diagnostics.
  - Reduced the wait time for MachineDeployment creation for AWS Spot instances. If AWS cannot allocate Spot instances because of insufficient capacity, retries begin sooner.
  - Added the `spec.kubelet.seccompDefault` parameter to the NodeGroup resource, allowing the RuntimeDefault seccomp profile to be enabled by default for all workloads without explicit seccomp settings.
- Reduced the probability of false restarts of the CSI Controller in all cloud provider modules during short delays by increasing the liveness probe timeout.
- Changes in the `cloud-provider-dvp` module:
  - Added support for live disk migration between StorageClass objects without stopping the virtual machine. Migration is performed automatically when the StorageClass changes.
  - The module configuration has been migrated to the unified ModuleConfig contract. Permanent and Ephemeral nodes are now configured in the same way through InstanceClass and NodeGroup, without a separate ProviderClusterConfiguration. This simplifies cluster configuration, enables schema versioning and conversion without breaking compatibility, and prepares cloud providers for an independent release cycle outside the DP core. For details, refer to “Cluster and Infrastructure” in the DP FAQ.
  - Fixed an issue that could leave LoadBalancer Services in the `Pending` state without creating the parent load balancer. Improved conflict handling during ServiceWithHealthchecks updates and load balancer status synchronization.
  - StorageClass objects created through Cluster API clusters now always use the `WaitForFirstConsumer` mode regardless of the parent cluster configuration. During updates, managed StorageClass objects with an incompatible `volumeBindingMode` are also recreated automatically. This eliminates an issue that could block cluster updates and synchronization because of the immutable `volumeBindingMode` field.
- The `cloud-provider-openstack`, `cloud-provider-huaweicloud`, `cloud-provider-vsphere`, and `cloud-provider-vcd` modules now use OpenTofu instead of Terraform. This makes it possible to use an open infrastructure platform compatible with Terraform, reduces dependency on Terraform licensing changes, and aligns these providers with other cloud provider modules that already use OpenTofu.
- The OpenStack and Yandex Cloud providers now use Cluster API, the industry-standard framework for machine management in Kubernetes. To ensure a smooth transition, the legacy mode based on Machine Controller Manager (MCM) remains supported. Existing node groups keep running on MCM, while new ones are created on Cluster API.
- Improved virtual machine deletion handling in the `cloud-provider-huaweicloud` module, increasing resource cleanup reliability and preventing false deletion errors. Virtual machines that remain in the deleting state for more than 24 hours are now marked with the corresponding status, simplifying diagnostics.
- Updated the CSI plugin and CSI Syncer versions in the `cloud-provider-vsphere` module, and fixed known vulnerabilities in Cloud Controller Manager, the CSI plugin, and Terraform Manager. This improves integration security and ensures the use of up-to-date component versions.
- Added support for the `vcd.cpi.flant.com/load-balancer-ip` annotation for LoadBalancer Services in the `cloud-provider-vcd` module. It allows assigning a predefined external IP address to the load balancer. If both the annotation and the `spec.loadBalancerIP` field are specified, the annotation takes precedence.

Network subsystem

- Changes in the `istio` module:
  - Added support for Istio 1.27. Users now have access to the new features of the latest Istio release using an operatorless mode.
  - Expanded ambient mesh support. New settings provide more explicit control over ambient mesh, allow using waypoint proxies for Layer 7 traffic processing, and enable the HBONE protocol for encrypting traffic between pods without requiring a sidecar container to run in every pod.

- Changes in the `cni-cilium` module:
  - Changed the behavior of NetworkPolicy objects with the `ipBlock: 0.0.0.0/0` rule in the new module version. The rule now also applies to intra-cluster traffic.
  - Reduced CPU consumption by Cilium agents. The freed resources become available to user workloads.
- Changes in the `ingress-nginx` module:
  - Added support for IngressNginxController v1.15.
  - Updated the load calculation formula for Ingress NGINX Controller pods used for horizontal scaling of LoadBalancer, LoadBalancerWithProxyProtocol, and LoadBalancerWithSSLPassthrough inlets. The controller now adds and removes replicas in a timely manner, resulting in more stable behavior under load.
  - Updated the IngressNginxController API to v2. The `spec.resourcesRequests` resource consumption settings have been replaced with the extended `spec.resourcesManagement` settings, including support for resource limits. Compatibility with the v1 API is preserved automatically.
  - Implemented isolated configuration validation, making it possible to restore full NGINX configuration validation without reducing the security level.
- Added authorization for the Hubble web interface in the `cilium-hubble` module. Access to cluster network telemetry can now be restricted by user groups to improve security.
- In the `metallb` module, BGP mode configuration has been moved from ModuleConfig to the dedicated MetalLoadBalancerPool, MetalLoadBalancerBGPPeer, and MetalLoadBalancerConfiguration custom resources. The existing configuration will be migrated automatically, so no manual changes in the cluster are required. All new BGP settings must be configured through these resources.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.77.10

Fixes

- **[node-manager]** Reduce node-manager Helm release Secret size to avoid exceeding the 1 MiB etcd limit on clusters with many Static or Hybrid NodeGroups.

1.77.9

Know before update

- Pods using common/nginx-static (registry-proxy, basic-auth, deckhouse-tools, documentation) will be restarted.
- Static pods and DaemonSet pods of `kubernetes-api-proxy` will restart.
- The registry-packages-proxy pods will be restarted.
- bashible-apiserver no longer skips such ModuleSources, so their CA is added to the node configuration. Bashible will be rerun on all nodes.

Fixes

- **[candi]** Bump base images to v3.1.28.
- **[ci]** Add Fomichev to codeowners for candi/base_images.yml
- **[cloud-provider-azure]** Disable the node-ipam controller in the cloud-controller-manager to prevent duplicate pod CIDRs and conflicting routes.
- **[cloud-provider-openstack]** Fix OpenStackCluster rendering when network lists are missing or empty
- **[cloud-provider-vcd]** preserve VM not-found errors to allow orphaned Node cleanup
- **[cloud-provider-vcd]** optimize VM cache refresh and avoid blocking cache reads
- **[cloud-provider-yandex]** quote label values in generated MachineClass manifests
- **[common]** Update nginx in common/nginx-static to 1.30.7 to fix CVEs.
  Pods using common/nginx-static (registry-proxy, basic-auth, deckhouse-tools, documentation) will be restarted.
- **[deckhouse-controller]** Deckhouse no longer crash-loops on a module dm-verity device left incomplete by an interrupted `veritysetup open`; such a device is now removed automatically.
- **[deckhouse-controller]** Releases of a still-embedded module no longer raise manual approval alerts once the module has a deployed release.
- **[docs]** Add note for using Ubuntu 26.04.
- **[node-manager]** The links in the FAQ on cleaning up a static node now lead to the section of the

sds-replicated-volume FAQ on evicting DRBD resources from a node.
- **[node-manager]** Disable HTTP keep-alives in kubernetes-api-proxy `/readyz` probes so connections close immediately after each check.
  Static pods and DaemonSet pods of `kubernetes-api-proxy` will restart.
- **[node-manager]** Restore the default maxSurge behavior for NodeGroups with maxSurgePerZone set to 0.
- **[node-manager]** Fix `StaticInstance` getting stuck in a bootstrap loop after cleanup, and status patches failing on legacy conditions.
- **[node-manager]** Nodes trust the registry CA from a ModuleSource that has `dockerCfg` set again.
  bashible-apiserver no longer skips such ModuleSources, so their CA is added to the node configuration. Bashible will be rerun on all nodes.
- **[registry-packages-proxy]** Reject CSE node packages with untrusted manifest signatures, including requests through registry aliases and cached downloads.
  The registry-packages-proxy pods will be restarted.

1.77.8

Know before update

- Dex pods restart. A client of Dex that expects HTTP 429 on the refresh of a locked user, HTTP 500 for a deleted one, or `invalid_request` for a claimed or expired refresh token gets HTTP 400 `invalid_grant`. The password and token exchange grants and the sign-in page answer as before.
- Existing clusters are unaffected; the default zone set (ru-central1-a, ru-central1-b, ru-central1-d, ru-central1-e) is statically pinned. To use ru-central1-f, list it in `zones` instead of one of the default zones; the dropped zone’s subnet is replaced by a new subnet in ru-central1-f and the nodes in it are recreated.
- containerd restarts on all nodes. On nodes where the old AppArmor profile is already loaded, the fix applies after a node reboot.

Features

- **[cloud-provider-yandex]** Add ru-central1-f zone support via explicit zone selection with a four-zone limit.
- **[cloud-provider-yandex]** Add ru-central1-f zone support via explicit zone selection with a four-zone limit.
  Existing clusters are unaffected; the default zone set (ru-central1-a, ru-central1-b, ru-central1-d, ru-central1-e) is statically pinned. To use ru-central1-f, list it in `zones` instead of one of the default zones; the dropped zone’s subnet is replaced by a new subnet in ru-central1-f and the nodes in it are recreated.

Fixes

- **[admission-policy-engine]** Pods with a Localhost AppArmor profile set in `securityContext` are no longer denied by the Baseline policy.
- **[ci]** Pass the werf meta repo to the dev site and documentation deploy jobs, matching the builds.
- **[common]** Updated `golang.org/x/net`, `golang.org/x/text`, `google.golang.org/grpc` and `github.com/klauspost/compress` in the Kubernetes 1.31 CSI sidecars of CSE to fix CVEs.
- **[control-plane-manager]** Fixed ControlPlaneOperation failures on clusters whose root CAs have a non-default subject.
- **[dhctl]** The bootstrap terminal no longer shows glued lines, FAILED for bashible step attempts that recovered, or “Unhandled Error” from client-go.
- **[docs]** Fix highlighting the L2 menu on module pages in the documentation.
- **[registrypackages]** Fix AppArmor denying signals between processes of one container on Ubuntu kernels 7.0.0-41 and later.
  containerd restarts on all nodes. On nodes where the old AppArmor profile is already loaded, the fix applies after a node reboot.
- **[user-authn]** Dex answers the refresh of a locked user, of a user deleted from the local store or LDAP, and of a claimed or expired refresh token with `invalid_grant` (HTTP 400), so that applications and a Dex that signs in through this one send the user to sign in.
  Dex pods restart. A client of Dex that expects HTTP 429 on the refresh of a locked user, HTTP 500 for a deleted one, or `invalid_request` for a claimed or expired refresh token gets HTTP 400 `invalid_grant`. The password and token exchange grants and the sign-in page answer as before.

1.77.7

Know before update

- A cluster that uses `Proxy` or `Local` of the previous registry implementation, or `Direct` without the `registry` ModuleConfig, is not updated to DP 1.78, as before. This now also applies from the moment such a mode is set in the `deckhouse` ModuleConfig, before the switch starts. Migrate as

described in “Migrating from the previous implementation to the current one” in the registry documentation. Clusters already prepared for the migration are not affected.
- A node that joins while a registry-packages-proxy pod of the previous release is still running on a master waits until a pod of this release answers.
- Cilium agent pods will be restarted.
- Flannel pods will be restarted.
- On a NodeGroup with `spec.gpu`, a failing GPU check used to abort the bashible run and be retried forever, so the node never configured kubelet and never received updates from any other module. Such a node now converges and only its GPU stays unusable. If `/usr/bin/nvidia-container-runtime` is absent, containerd is now configured with `runc` as the default runtime instead of `nvidia`; on a node whose configuration previously selected `nvidia` this changes the containerd config and therefore restarts containerd. Previously such a node could not start any container at all.
- Pods of the kube-proxy, cni-cilium, cni-flannel, and cni-simple-bridge agent DaemonSets will be restarted because their cni-migration-init-checker init container image is updated.
- The controller and agent pods will be restarted.
- The registry in-cluster proxy and node services pods will be restarted.
- The terraform-manager pods will be restarted.
- istiod, istio-cni, ingress gateways, Kiali, api-proxy and metadata-exporter pods will be restarted.
- node-controller, capi-controller-manager, machine-controller-manager and bashible-apiserver do not run on a master tainted with a key from `global.modules.placement.customTolerationKeys`. Remove such taints from the master NodeGroup before the update.
- simple-bridge pods will be restarted.

Features

- **[docs]** Implement glossary with term linking.
- **[docs]** Added the “System requirements” page with deployment options and resource requirements for containers, virtual machines and combined workloads.
- **[node-manager]** Alert when a NodeGroup uses the deprecated built-in GPU support while the `gpu` module is disabled.

Fixes

- **[candi]** A GPU problem no longer blocks node convergence.
  On a NodeGroup with `spec.gpu`, a failing GPU check used to abort the bashible run and be retried forever, so the node never configured kubelet and never received updates from any other module. Such a node now converges and only its GPU stays unusable. If `/usr/bin/nvidia-container-runtime` is absent, containerd is now configured with `runc` as the default runtime instead of `nvidia`; on a node whose configuration previously selected `nvidia` this changes the containerd config and therefore restarts containerd. Previously such a node could not start any container at all.
- **[candi]** GPU compute capability check no longer requires `bc`.
- **[candi]** Persist `net.core.bpf_jit_harden` for GPU nodes in an own sysctl drop-in instead of editing distribution files that may be absent.
- **[candi]** bashible verifies the rpp-get binary against the image digest from its bundle before running it; a substituted bootstrap proxy on a master can no longer hand a binary to the nodes.
  A node that joins while a registry-packages-proxy pod of the previous release is still running on a master waits until a pod of this release answers.
- **[candi]** Fixed an intermittent bootstrap hang where kube-apiserver never started because kubelet read its manifest while it was being written.
- **[candi]** A new cloud node sets up the provider network before it installs bootstrap packages, so it can download them from a master in another zone.
- **[chrony]** Fixed known vulnerabilities in chrony-exporter.
- **[cloud-provider-dvp]** Fixes DVP CSI staging so volumes with filesystems detected by Kubernetes SafeFormatAndMount are preserved and unformatted volumes are formatted without using lsblk.
- **[cloud-provider-dvp]** Creates ModuleConfig/cloud-provider-dvp automatically when it is missing.
- **[cni-cilium]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so Cilium agents start on Deckhouse Engine nodes.
  Cilium agent pods will be restarted.
- **[cni-flannel]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so Flannel pods start on Deckhouse Engine nodes.
  Flannel pods will be restarted.
- **[cni-simple-bridge]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so simple-bridge pods start on Deckhouse Engine nodes.
  simple-bridge pods will be restarted.
- **[common]** Fixed known vulnerabilities in the cni-migration-controller and cni-migration-init-checker images.
  Pods of the kube-proxy, cni-cilium, cni-flannel, and cni-simple-bridge agent DaemonSets will be restarted because their cni-migration-init-checker init container image is updated.
- **[common]** added mountpoint in ipv4-options-guard
- **[common]** Updated `golang.org/x/net`, `golang.org/x/text`, `google.golang.org/grpc` and

`github.com/klauspost/compress` in CSI sidecars to fix CVEs.
- **[control-plane-manager]** Preserve custom TLS certificates when enabling a default Gateway.
- **[control-plane-manager]** The migration hook for the api listener is added.
- **[deckhouse]** Fixed a memory leak in webhook-handler, which kept the kubernetes monitors of replaced hooks running after every hook reload.
- **[deckhouse-controller]** Fixed image pulls for modules served by ModulePullOverride when another source updates its registry or the override is not ready during Deckhouse startup.
- **[dhctl]** Fixed known vulnerabilities in dhctl dependencies.
- **[docs]** Move the language switch from the header to the footer
- **[documentation]** Add the language switch to the footer.
- **[istio]** Fixed CVEs in the istio module images (go-jose, golang.org/x/net).
  istiod, istio-cni, ingress gateways, Kiali, api-proxy and metadata-exporter pods will be restarted.
- **[local-path-provisioner]** Updated `golang.org/x/net`, `golang.org/x/text` and `github.com/klauspost/compress` to fix CVEs.
- **[metallb]** Fixed BGP migration to preserve pool allocation flags and allow users to manage migrated resources after switching ModuleConfig to version 3.
- **[node-manager]** Master controllers leave an unreachable master after about 100 seconds and are not evicted from a single master.
- **[node-manager]** Master controllers no longer tolerate keys from `customTolerationKeys`, so a key that repeats a built-in one no longer blocks pod updates.
  node-controller, capi-controller-manager, machine-controller-manager and bashible-apiserver do not run on a master tainted with a key from `global.modules.placement.customTolerationKeys`. Remove such taints from the master NodeGroup before the update.
- **[node-manager]** A node-manager release no longer waits for capi-controller-manager when neither capi nor its webhooks change.
- **[node-manager]** Fixed known vulnerabilities in the bashible-apiserver, node-controller, node-group-exporter, kubernetes-api-proxy, d8-shutdown-inhibitor and capi-controller-manager images.
- **[node-manager]** Fixed constant NodeGroup status updates by node-controller on every Node update.
- **[registry]** Fixed known vulnerabilities in the docker-auth image.
  The registry in-cluster proxy and node services pods will be restarted.
- **[registry]** Fixed known vulnerabilities in the nodeservices-manager, docker-auth, mirrorer and syncer images.
- **[registry]** The update to DP 1.78 is held as soon as `registry.mode` `Proxy` or `Local`, or a switch between `Direct` and `Unmanaged`, is set in the `deckhouse` ModuleConfig, and the message of the held release lists the migration steps for each mode.
  A cluster that uses `Proxy` or `Local` of the previous registry implementation, or `Direct` without the `registry` ModuleConfig, is not updated to DP 1.78, as before. This now also applies from the moment such a mode is set in the `deckhouse` ModuleConfig, before the switch starts. Migrate as described in “Migrating from the previous implementation to the current one” in the registry documentation. Clusters already prepared for the migration are not affected.
- **[registry-packages-proxy]** The bootstrap port serves the manifest and the layer of the rpp-get image, so a node can verify the binary it installs.
- **[registry-packages-proxy]** Fixed known vulnerabilities in registry-packages-proxy.
- **[registrypackages]** Fixed known vulnerabilities in the cfssl and d8-bcheck packages.
- **[registrypackages]** Bump golang.org/x/net to v0.62.0 in d8-bcheck to fix CVE-2026-78661 and CVE-2026-97045.
- **[registrypackages]** Fixed known vulnerabilities in the rpp-get package.
- **[registrypackages]** Fixed known vulnerabilities in auth_server and registry of the docker-registry package.
- **[registrypackages]** rpp-get tries every registry-packages-proxy endpoint before an attempt fails and retries connection timeouts, so a node that reaches only the master of its own zone during bootstrap joins the cluster.
- **[service-with-healthchecks]** Go dependencies are updated.
  The controller and agent pods will be restarted.
- **[terraform-manager]** Fixed known vulnerabilities in opentofu.
  The terraform-manager pods will be restarted.
- **[terraform-manager]** Fixed known vulnerabilities in opentofu and in the terraform-manager-dvp image.
- **[tools]** The e2e directories of modules no longer reach the Deckhouse image.
- **[user-authn]** Default DKP gateway requirement is removed from dex-authenticator template.

1.77.6

Know before update

- Applying a discovered address restarts the Deckhouse Pod once.
- Cilium agent pods will be restarted. With DSR, the DSR IPv4 option is stripped before the packet reaches the pod.
- Control plane components (kube-apiserver, kube-controller-manager, kube-scheduler, etcd) and kubelet will be restarted.

- Flannel pods will be restarted.
- In ambient mode with Istio 1.27 or later, on nodes with legacy iptables, `istio-cni-node` switches from nftables to legacy iptables. Pods that were added to the mesh before the update keep their nftables rules. Restart these pods after the update. Otherwise, removing such a pod from the mesh without a restart leaves its old redirect rules in place, and the pod loses network connectivity.
- The controller and agent pods will be restared.
- The metallb components (controller, speaker, l2lb) will restart after the update.
- containerd will be restarted on all nodes.
- kube-proxy pods will be restarted.
- simple-bridge pods will be restarted.

Features

- **[cni-cilium]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new clusters, `Audit` for existing ones).
  Cilium agent pods will be restarted. With DSR, the DSR IPv4 option is stripped before the packet reaches the pod.
- **[cni-flannel]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new clusters, `Audit` for existing ones).
  Flannel pods will be restarted.
- **[cni-simple-bridge]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new clusters, `Audit` for existing ones).
  simple-bridge pods will be restarted.
- **[docs]** Add static-routing-manager architecture module section.
- **[docs]** Add cni-flannel and cni-simple-bridge modules architecture section.

Fixes

- **[admission-policy-engine]** Fixed CVE-2026-116585, CVE-2026-116586 and CVE-2026-116587 in `github.com/go-jose/go-jose/v4` and CVE-2026-78661 and CVE-2026-97045 in `golang.org/x/net`.
- **[admission-policy-engine]** Switch ratify to go-jose v4 to fix CVE-2026-116585, CVE-2026-116586, CVE-2026-116587.
- **[admission-policy-engine]** An unavailable Gatekeeper no longer blocks system components in namespaces that opted into security policy checks, which could deadlock the cluster together with the CNI agent.
- **[cert-manager]** Bump x/net to v0.62.0 and go-jose to v4.1.6 to fix CVE-2026-78661, CVE-2026-97045, CVE-2026-116585, CVE-2026-116586, CVE-2026-116587.
- **[ci]** Svace stages are rebuilt in every svace pipeline.
- **[cloud-provider-dvp]** Fixes x/net, x/text and kin-openapi CVEs in all module images.
- **[cloud-provider-openstack]** Fixes creation of nodes with `rootDiskSize` when the node availability zone does not exist in Cinder.
- **[cloud-provider-openstack]** fix CloudEphemeral node creation in non-default availability zones via automatic Cinder-AZ discovery and new OpenStackInstanceClass.volumeTypeMap field
- **[cni-cilium]** Fixed file descriptor, connection and lock leaks in Cilium found by static analysis.
- **[common]** Bump x/net in kube-rbac-proxy to v0.62.0 to fix CVE-2026-78661 and CVE-2026-97045.
- **[control-plane-manager]** The basic audit policy records an exec into a pod (`pods/exec`) in all namespaces again, as it did before Deckhouse 1.77.
- **[control-plane-manager]** Fixed a goroutine leak in the helm release deprecation scan when listing releases fails.
- **[control-plane-manager]** Updated go-jose, x/net and klauspost/compress in control plane components to fix known vulnerabilities.
  Control plane components (kube-apiserver, kube-controller-manager, kube-scheduler, etcd) and kubelet will be restarted.
- **[deckhouse]** Resolve `registry.d8-system.svc` to a node running the in-cluster registry proxy during bootstrap, instead of always to `127.0.0.1`.
  Applying a discovered address restarts the Deckhouse Pod once.
- **[deckhouse]** Update runtime dependencies and document unaffected vulnerability paths.
- **[deckhouse]** The `d8-main` Telemetry no longer references the `deckhouse-tracing` provider when the module doesn’t register it.
- **[deckhouse-controller]** Prevent module hooks from running with unavailable snapshots while waiting for Deckhouse to restart after a module update
- **[deckhouse-controller]** A pending module release no longer loses its Kubernetes, Deckhouse and module requirements, so an update that the cluster does not satisfy stays pending instead of disabling the module.
- **[dhctl]** Keep the SSH host list and master readiness checks in sync with the infrastructure state during master converge.
- **[dhctl]** Bump go-jose to v4.1.6 to fix CVE-2026-116585, CVE-2026-116586, CVE-2026-116587.

- **[docs]** Fix docs build config errors in werf includes.
- **[documentation]** Fix HTML parser vulnerabilities in docs-builder by updating golang.org/x/net to v0.62.0.
- **[go_lib]** Package proxy streams the full package body when the cache skips writing it.
- **[istio]** Fixed ambient mode on nodes with legacy iptables, where `istio-cni-node` crashed on Istio 1.25 and used nftables instead of legacy iptables on Istio 1.27 and later.
  In ambient mode with Istio 1.27 or later, on nodes with legacy iptables, `istio-cni-node` switches from nftables to legacy iptables. Pods that were added to the mesh before the update keep their nftables rules. Restart these pods after the update. Otherwise, removing such a pod from the mesh without a restart leaves its old redirect rules in place, and the pod loses network connectivity.
- **[kube-proxy]** Fixed known vulnerabilities in the kube-proxy images.
  kube-proxy pods will be restarted.
- **[local-path-provisioner]** `config.json` of the provisioner is serialized from a structure, and the CRD accepts in `spec.path` only an absolute path without quotes, backslashes, whitespace and control characters. A crafted path can no longer add manifests to the module release.
- **[metallb]** Fixed CVE-2026-78661 and CVE-2026-97045 in the metallb and l2lb images.
  The metallb components (controller, speaker, l2lb) will restart after the update.
- **[multitenancy-manager]** Fixed CVE-2026-116585, CVE-2026-116586 and CVE-2026-116587 in `github.com/go-jose/go-jose/v4` and CVE-2026-78661 and CVE-2026-97045 in `golang.org/x/net`.
- **[node-manager]** kube-rbac-proxy in capi-controller-manager and node-controller no longer listens on all addresses of a node that has no addresses yet, so it does not take the controller's port.
- **[node-manager]** Fixed known vulnerabilities in the integrity-controller and integrity-containerd-configurator images (CSE).
- **[node-manager]** Fixes x/net CVEs in the caps-controller-manager image.
- **[registry]** Require brackets around IPv6 upstream registry URL hosts and accept bracketed addresses with or without a port.
- **[registry-packages-proxy]** Fixed registry-packages-proxy hanging after a cache error and serving corrupted packages, which blocked node updates.
- **[registry-packages-proxy]** Bump go-jose to v4.1.6 to fix CVE-2026-116585, CVE-2026-116586, CVE-2026-116587.
- **[registrypackages]** Updated go-jose, x/net and klauspost/compress in containerd, runc and crictl to fix known vulnerabilities.
  containerd will be restarted on all nodes.
- **[registrypackages]** Support rebuilding legacy image layers and preserve verity data in CSE.
- **[service-with-healthchecks]** Status updates were refactored.
  The controller and agent pods will be restared.
- **[user-authn]** Local users keep the passwords they changed in Dex when the cluster updates from 1.76.
- **[user-authn]** Fixed CVE-2026-116585, CVE-2026-116586 and CVE-2026-116587 in `github.com/go-jose/go-jose/v4` and CVE-2026-78661 and CVE-2026-97045 in `golang.org/x/net`.
- **[user-authn]** Bump go-oidc in dex-authenticator to v3.11.0 to fix CVE-2026-116585, CVE-2026-116586, CVE-2026-116587.
- **[user-authz]** Fixed CVE-2026-78661 and CVE-2026-97045 in `golang.org/x/net`.
- **[vertical-pod-autoscaler]** Fix a panic in vpa-updater and vpa-admission-controller when a pod with a container without resources is capped to a LimitRange of type Pod.

1.77.5

Know before update

- After a reset, the previous password cannot be reused when passwordHistoryLimit is set; the current password cannot be set again through the change form, and a lowered passwordHistoryLimit applies at the next password change.
- Cilium agent and operator pods will be restarted.
- Control plane components and kubelet will be restarted.
- CoreDNS pods will be restarted.
- Node-local-dns safe updater will be restarted.
- TLS connections from nodes to hosts whose certificates chain to `emSign ECC Root CA - G3` or Comodo `AAA Certificate Services` (eMudhra / emSign issued) will fail. Reissue such certificates from another CA before the update.
- The `cert-manager` controller pod restarts.
- The controller and agent pods will be restarted.
- cni-flannel pods will be restarted.
- kube-proxy pods will be restarted.
- network-policy-engine pods will be restarted.
- node-local-dns pods will be restarted.

Features

- **[control-plane-manager]** The basic audit policy logs Pod status updates from kubelet with the request body, so container terminations and their reasons get into the audit log.
- **[deckhouse]** Migrated package release management to nelm v2 while keeping the v1 install and delete behaviour.
- **[user-authn]** DexAuthenticator logins, logouts, failed and denied logins, CSRF mismatches and failed session refreshes are delivered as security events when the `security-events-manager` module is enabled.

Fixes

- **[admission-policy-engine]** Fixed CVE-2026-114437 in `github.com/go-chi/chi/v5` and CVE-2026-63209 in `github.com/klauspost/compress`.
- **[candi]** The first master bootstrap waits for `kube-apiserver` readiness instead of failing the control-plane step on `connection refused`.
- **[cert-manager]** The metrics proxy of the controller is no longer restarted by its liveness probe while its node is short of CPU.
  The `cert-manager` controller pod restarts.
- **[cilium-hubble]** Fixed a known vulnerability in the Hubble relay image.
- **[cloud-provider-dvp]** Fixed deletion of load balancers backed by ServiceWithHealthchecks, which failed because the child Service is protected by the `heritage - deckhouse` label in the parent cluster.
- **[cloud-provider-huaweicloud]** Recreated the etcd disk when a master VM is recreated.
- **[cloud-provider-openstack]** CAPI now mirrors MCM — `configDrive` honors `externalNetworkDHCP`, and additional networks always produce ports
- **[cloud-provider-vcd]** Fixes the module getting stuck in Error on upgrade from 1.76 because of the immutable selector of the `capcd-controller-manager` Deployment.
- **[cni-cilium]** Fixed a known vulnerability in the Cilium images.
  Cilium agent and operator pods will be restarted.
- **[cni-flannel]** Fixed known vulnerabilities in the flanneld image.
  cni-flannel pods will be restarted.
- **[control-plane-manager]** Updated Go dependencies of Kubernetes control plane components to fix known vulnerabilities.
  Control plane components and kubelet will be restarted.
- **[control-plane-manager]** The cluster-wide controllers of `control-plane-manager` run in a single pod regardless of the `highAvailability` setting or an etcd arbiter.
- **[control-plane-manager]** Control plane components are no longer restarted in parallel when a node is NotReady or after a certificate renewal.
- **[control-plane-manager]** Fixed a Deckhouse crash caused by a helm release `Secret` or `ConfigMap` with too short release data.
- **[control-plane-manager]** removed unused executable files from the etcd-backup image
- **[control-plane-manager]** In CSE, `availableVersions` in `d8-cluster-kubernetes` lists only Kubernetes versions that can be set.
- **[deckhouse]** Restore ClusterAdmin read access to ModuleSettingsDefinition so the console can show module settings again.
- **[deckhouse]** Show the Terminating phase while package repositories are being deleted
- **[deckhouse]** Update d8 CLI and scoped runtime VEX exceptions for known vulnerabilities.
- **[deckhouse-controller]** A module enabled in the bundle is no longer disabled when its ModuleConfig does not set `enabled`.
- **[deckhouse-controller]** Module sources now scan the registry every 30 minutes by default, reducing registry load
- **[istio]** added limit paralle in common images
- **[istio]** Fixed CVE-2026-63209 in istio images.
- **[istio]** Fixed istio module installation with nelm v2 when Istio is managed by the istio operator (versions below 1.27).
- **[istio]** Fixed `alliance-healthcheck` keeping the sidecar of the previous Istio version after a `globalVersion` change.
- **[kube-dns]** Fixed a known vulnerability in the CoreDNS image.
  CoreDNS pods will be restarted.
- **[kube-proxy]** Fixed known vulnerabilities in the kube-proxy init-container image.
  kube-proxy pods will be restarted.
- **[kube-proxy]** Fixed known vulnerabilities in the kube-proxy images.
  kube-proxy pods will be restarted.
- **[multitenancy-manager]** Fixed CVE-2026-63209 in `github.com/klauspost/compress`.
- **[network-policy-engine]** Fixed known vulnerabilities in the kube-router image.
  network-policy-engine pods will be restarted.
- **[node-local-dns]** Fixed DNS errors during node-local-dns pod restarts in clusters with Cilium.
- **[node-local-dns]** Fixed a known vulnerability in the node-local-dns CoreDNS image.
  node-local-dns pods will be restarted.
- **[node-local-dns]** Node-local-dns safe updater logic was updated to match the Kubernetes logic.
  Node-local-dns safe updater will be restarted.
- **[openvpn]** Fixed known vulnerabilities in the ovpn-admin and easyrsa-migrator images.
- **[registry]** In the `Proxy` mode, reuse upstream auth challenges and tokens instead of requesting them before every request to the upstream registry.

- **[registry]** Fix the registry static pod stuck in `ImagePullBackOff` after a registry image update on a single master in the `Proxy` and `Local` modes.
- **[registrypackages]** Nodes stop trusting `emSign ECC Root CA - G3` and Comodo `AAA Certificate Services`, the roots behind the compromised eMudhra intermediate CAs. TLS connections from nodes to hosts whose certificates chain to `emSign ECC Root CA - G3` or Comodo `AAA Certificate Services` (eMudhra / emSign issued) will fail. Reissue such certificates from another CA before the update.
- **[registrypackages]** Update containerdv2 patch
- **[service-with-healthchecks]** CVEs were fixed.
  The controller and agent pods will be restarted.
- **[user-authn]** dex-authenticator no longer writes the request headers, cookies included, to the log when the CSRF cookie is missing.
- **[user-authn]** A password reset of a local user keeps the replaced password in the password history, and the password change form refuses the current password in every case.
  After a reset, the previous password cannot be reused when passwordHistoryLimit is set; the current password cannot be set again through the change form, and a lowered passwordHistoryLimit applies at the next password change.

1.77.4

Know before update

- Cilium agent and operator pods will be restarted.
- CoreDNS pods will be restarted.
- NodeGroups that already have preemptible: true under the CAPI engine keep their current regular VMs until the next rollout (any rollout-triggering InstanceClass change or manual-rollout-id); after that rollout their nodes become preemptible VMs and are rotated every ~20h.
- Upgrade does not recreate any node. NodeGroups that already have preemptible: true under the CAPI engine keep their current regular VMs; Machines created after the upgrade (scale-up, replacement) are preemptible, so such NodeGroups run mixed VMs until a full rollout (any rollout-triggering InstanceClass change or the manual-rollout-id annotation). Preemptible VMs are rotated every ~20h.
- metallb pods will be restarted.
- node-local-dns pods will be restarted.

Features

- **[control-plane-manager]** The `etcd` dashboard is reworked to find the cause of slow `etcd` - disk, network, leader, `kube-apiserver` load or page cache.

Fixes

- **[chrony]** Bump golang.org/x/net in chrony_exporter to fix CVE-2026-78669.
- **[cilium-hubble]** Fixed known vulnerabilities in the Hubble relay and UI images.
- **[cloud-provider-aws]** Allow bootstrapping clusters in AWS regions unknown to the bundled terraform-provider-aws, such as `ap-southeast-6`.
- **[cloud-provider-dvp]** A DVP cluster migrated to the new provider configuration format keeps its `networkPolicy - None` instead of silently switching to `Isolated`.
- **[cloud-provider-dvp]** Clusters with `networkPolicy - None` in the provider configuration no longer get NetworkPolicies created for LoadBalancer Services.
- **[cloud-provider-yandex]** Preemptible nodes are created as preemptible VMs and rotated before the 24h limit when a NodeGroup is managed by the CAPI engine.
  Upgrade does not recreate any node. NodeGroups that already have preemptible: true under the CAPI engine keep their current regular VMs; Machines created after the upgrade (scale-up, replacement) are preemptible, so such NodeGroups run mixed VMs until a full rollout (any rollout-triggering InstanceClass change or the manual-rollout-id annotation). Preemptible VMs are rotated every ~20h.
- **[cloud-provider-yandex]** Preemptible nodes are created as preemptible VMs and rotated before the 24h limit when a NodeGroup is managed by the CAPI engine.
  NodeGroups that already have preemptible: true under the CAPI engine keep their current regular VMs until the next rollout (any rollout-triggering InstanceClass change or manual-rollout-id); after that rollout their nodes become preemptible VMs and are rotated every ~20h.
- **[cni-cilium]** Fixed known vulnerabilities in the Cilium images.
  Cilium agent and operator pods will be restarted.
- **[common]** Fixed known vulnerabilities in the CNI migration images.
- **[deckhouse]** Bump containerd to v2.2.10 and dependencies to fix CVEs
- **[dhctl]** Bump golang.org/x/net to fix CVE-2026-78669.
- **[docs]** Update terminology in the documentation and SEO improvements.
- **[documentation]** Update documentation building rules.
- **[istio]** Added VEX for CVE-2026-85747 in pilot and proxyv2 images.
- **[kube-dns]** Fixed known vulnerabilities in the CoreDNS and sts-pods-hosts-appender images.
  CoreDNS pods will be restarted.
- **[metallb]** Fixed known vulnerabilities in the MetalLB images.

metallb pods will be restarted.
- **[node-local-dns]** Fixed known vulnerabilities in the node-local-dns images.
  node-local-dns pods will be restarted.
- **[node-manager]** Updates otel, grpc and x/crypto dependencies of node-controller to fix CVEs.
- **[node-manager]** fixed private build
- **[node-manager]** Bump golang.org/x/net to fix CVE-2026-78669, CVE-2026-78663, CVE-2026-97032.
- **[registry]** Bump golang.org/x/net in nodeservices-manager to fix CVE-2026-78669.
- **[registry-packages-proxy]** Bump golang.org/x/net to fix CVE-2026-78669.
- **[registrypackages]** Fixed known vulnerabilities in the kubernetes-cni package.
- **[registrypackages]** Bump golang.org/x/net, golang.org/x/crypto and OpenTelemetry in rpp-get and the docker-registry auth_server to fix CVE-2026-78669, CVE-2026-56855, CVE-2026-81870.
- **[terraform-manager]** Bump golang.org/x/net in opentofu to fix CVE-2026-78669.
- **[vertical-pod-autoscaler]** The admission-controller webhook now serves HTTP/1.1 only and no longer exposes

1.77.3

Know before update

- Before, such an entry stopped the webhook from reloading the rules, so subjects of rules added later, and every subject after a restart of the webhook, were not limited to their namespaces. Subjects of a rule whose only entry is invalid now get no namespace through the entries of that rule.
- Bumping `deckhouse lib helm` from `1.72.11` to `1.72.24` also drops the `drbd.linbit.com` taint tolerations from `helm_lib_tolerations`, so pods rendered through it no longer tolerate `lost-quorum`, `force-io-error` and `ignore-fail-over`, and aligns the VPA `maxAllowed` memory to the recommender’s 64Mi quantum.
- Exec and attach requests in user namespaces reach Gatekeeper again, as in Deckhouse 1.73 and 1.74. Every constraint that matches them is evaluated again, including a constraint without `match.kinds` or with `kinds: ["*"]` whose policy does not check `input.review.operation`. Before updating, check that such constraints do not deny `kubectl exec` and `kubectl attach` for users who need them.
- In a cluster with the `proxy` parameter of `ClusterConfiguration` set, the `cert-manager` controller sends requests to the ACME server, HashiCorp Vault and the DNS provider APIs through the proxy server. The `HTTP-01` self-check also requests the domain through the proxy server. If the proxy server cannot reach the issuer or the public address of the cluster, add these addresses to `noProxy`.
- In single-replica installations a new controller pod reconciles projects only after the previous pod has stopped and released the lease. If the previous pod is lost without a shutdown, for example together with its node, the new pod starts reconciling projects 60 to about 78 seconds after it starts.
- Machines are not recreated on update. Machines created before the update keep the standard network type and no labels until the NodeGroup is rolled out (change a rolling InstanceClass field or set the `manual-rollout-id` annotation on the NodeGroup). Machines stuck in provisioning because of `additionalSubnets` become ready without recreation.
- Such identities can write ClusterAuthorizationRules, Users and Groups that carry `SuperAdmin` or `cluster-admin`. A ClusterAuthorizationRule gives this right only without `namespaceSelector` and `limitNamespaces` and, with multi-tenancy on, only with `allowAccessToSystemNamespaces: true`, even if it grants `cluster-admin`.
- The cloud MachineHealthCheck is updated in existing clusters. A node in Ready=False is replaced after 10 minutes instead of 5 (Ready=Unknown still after 5), and remediation stops while more than 50% of the cluster’s CAPI machines are unhealthy.
- The controller and agents will be restarted.
- The grant webhooks are now removed together with the module. On a cluster where `multitenancy-manager` is already disabled at the update, the `cluster-objects-grants-defaulting` and `cluster-objects-grants-validator` webhook configurations left by earlier versions stay, and writes to grantable resources in project namespaces (for example, removing a PVC finalizer) keep failing with `service "multitenancy-manager" not found`. Delete them by hand: `d8 k delete mutatingwebhookconfiguration cluster-objects-grants-defaulting` `d8 k delete validatingwebhookconfiguration cluster-objects-grants-validator` Nothing has to be done on clusters where the module is enabled.

Features

- **[admission-policy-engine]** Report the availability of the module components with alerts.
- **[cert-manager]** Add the `highAvailability` parameter to control the HA mode of the module independently of the global setting.
- **[cert-manager]** `cert-manager` uses the cluster-wide proxy server from the `proxy` parameter of `ClusterConfiguration`.
  In a cluster with the `proxy` parameter of `ClusterConfiguration` set, the `cert-manager` controller sends requests to the ACME server, HashiCorp Vault and the DNS provider APIs through the proxy server. The `HTTP-01` self-check also requests the domain through the proxy server. If

the proxy server cannot reach the issuer or the public address of the cluster, add these addresses to `noProxy`.
- **[cloud-provider-dvp]** Makes CCM create NPs on every lb creation while using networkPolicy - Restricted
- **[cni-cilium]** Added an alert and a release requirement that block the Cilium update while CiliumNetworkPolicy or CiliumClusterwideNetworkPolicy objects use the `fromRequires` or `toRequires` fields.
- **[deckhouse]** ModuleSource scans skip unchanged release channels using a cheap registry HEAD request instead of pulling the whole release image every cycle.
- **[deckhouse-controller]** Added the update settings of applications: the `spec.update` field of an Application and the global `applications.update` parameter. Deckhouse merges them with the update settings of the `deckhouse` module but doesn't apply them to application updates yet.
- **[docs]** Add user-authz module architecture section.
- **[docs]** Add service-with-healthchecks module architecture section.
- **[documentation]** The in-cluster documentation now includes the guides.
- **[documentation]** Documentation updates for DP CSE 1.77.
- **[multitenancy-manager]** A new alert warns about a user project template named `simple`, which the built-in template of the same name overwrites after the update.
- **[user-authz]** The new `D8UserAuthzLegacyRBACv2CapabilityBindingFound` alert lists the bindings to built-in capabilities that the next minor release renames without aliases, and names the new name of each capability.
- **[user-authz]** The new `D8UserAuthzLegacyRBACv2CustomCapabilityFound` alert lists custom capabilities that extend the namespace roles only through labels the next role scheme does not select.
- **[user-authz]** New `D8UserAuthzLimitNamespacesEntryNeedsRewrite` and `D8UserAuthzLimitNamespacesEntryInvalid` alerts report ClusterAuthorizationRule `limitNamespaces` entries that are anchored only in part or are not valid regular expressions.

Fixes

- **[admission-policy-engine]** Fixed Gatekeeper policies for `kubectl exec` and `kubectl attach` being skipped in user namespaces.
  Exec and attach requests in user namespaces reach Gatekeeper again, as in Deckhouse 1.73 and 1.74. Every constraint that matches them is evaluated again, including a constraint without `match.kinds` or with `kinds: ["*"]` whose policy does not check `input.review.operation`. Before updating, check that such constraints do not deny `kubectl exec` and `kubectl attach` for users who need them.
- **[admission-policy-engine]** Disabling the module deletes the webhook configurations before Gatekeeper, so the uninstall no longer blocks `exec` and the Deckhouse queue.
- **[admission-policy-engine]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net` and CVE-2026-56851 in `golang.org/x/text`.
- **[candi]** Restored the lost `kubelet-disable-k-panic-check` patch for k8s 1.35 and 1.36, so kubelet no longer enforces `kernel.panic=10` on these versions.
- **[candi]** New nodes reboot on the first bashible run again when a step requests it. Only the first master skips this reboot.
- **[cert-manager]** The webhook keeps serving requests during a rollout and during a rotation of its CA, and the controller waits for the API server at startup instead of restarting.
- **[cert-manager]** The `spec.certificateOwnerRef` field of a certificate no longer causes an endless Secret update loop when it differs from the global owner reference setting that `cleanupOrphanSecrets` enables in the `CertManager` HTTPS mode, and is applied to an already issued Secret.
- **[cert-manager]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net`.
- **[chrony]** Bump golang.org/x/crypto in chrony_exporter to fix CVE-2026-56855, CVE-2026-78662.
- **[cloud-provider-dvp]** The CSI driver reports a volume as detached from a node only after the disk is released by the node virtual machine, so with virtualization that keeps the VM listed on the disk until it is released, the volume is not attached to two nodes at once.
- **[cloud-provider-dvp]** The CSI driver reports the volume limit of a node, so the scheduler does not place more volumes on a node than its virtual machine can attach.
- **[cloud-provider-dvp]** Stop infinite check/converge loop caused by the DVP controller annotation on the cloud-init Secret.
- **[cloud-provider-vsphere]** Fixed master node creation when the datastore is set as a path with parent folders.
- **[cloud-provider-yandex]** Restored `additionalSubnets`, `networkType` and VM labels (`nodes.parameters.labels`, `additionalLabels`) for CAPI-managed Yandex nodes.
  Machines are not recreated on update. Machines created before the update keep the standard network type and no labels until the NodeGroup is rolled out (change a rolling InstanceClass field or set the `manual-rollout-id` annotation on the NodeGroup). Machines stuck in provisioning because of `additionalSubnets` become ready without recreation.
- **[cloud-provider-zvirt]** VMs are now actually removed from zVirt on NodeGroup scale-down and cluster destroy
- **[common]** Fixed a `kube-apiserver` panic on JSON Patch requests whose `add`/`replace` of the

whole document or `test` operation lacks the `value` field; such requests are now rejected with a validation error.
- **[common]** kube-apiserver returns a single short validation error for an invalid `fieldManager` instead of an error for every invalid character.
- **[common]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net` in kube-rbac-proxy.
- **[control-plane-manager]** Keep `resource.k8s.io/v1beta1` enabled in `kube-apiserver` 1.34.
- **[deckhouse]** Fixed CVEs in d8 CLI dependencies by updating d8 to v0.33.25.
- **[deckhouse]** Restore kubectl plugin lookup in d8 k and d8 kubectl
- **[deckhouse]** Update runtime libraries and VEX exceptions for known vulnerabilities
- **[deckhouse-controller]** Deleting an Application now also deletes objects with the `werf.io/ownership: anyone` or `werf.io/resource-policy` annotation, such as Jobs deployed only on install, if the package lists their kinds in `orphanResources`. Only `helm.sh/resource-policy: keep` protects an object.
- **[descheduler]** Fixed CVE-2026-81870 in descheduler by bumping OpenTelemetry Go modules to v1.45.0.
- **[dhctl]** Preserve existing SSH keys when switching to the temporary converge user.
- **[dhctl]** Destroy and abort of a static cluster now fail instead of reporting success when a control-plane node was not cleaned because of an SSH error or cleanup timeout.
- **[dhctl]** Bump golang.org/x/crypto and OpenTelemetry to fix CVE-2026-56855, CVE-2026-78662, CVE-2026-81870, CVE-2026-81871, CVE-2026-81872.
- **[docs]** Correct typo in managed-rabbit key in modules library.
- **[docs]** Improve documentation clarity and fix typos.
- **[docs]** Fixed tab switching on pages with several tab sets of the same name and opening hidden tabs by an anchor link.
- **[docs]** Enhance tab activation logic in the documentation.
- **[docs]** Enhance tab shortcode functionality.
- **[docs]** Remove raw tags from usage documentation.
- **[docs]** Show expand icon on non-collapsible top-level CRD parameters in the documentation.
- **[docs]** Added experimental module enable note to the lifecycle badge.
- **[docs]** Added registry module port 5001 to Deckhouse ports documentation
- **[docs]** Rendered all versions from oss.yaml versions array.
- **[docs]** Update hugo to 0.168.0 in docs-builder
- **[docs]** Page with DNS names full list was added in to reference.
- **[helm_lib]** Size the resources of a module whose `resourcesManagement` names a fractional quantity, instead of rendering a zero limit.
  Bumping `deckhouse_lib_helm` from `1.72.11` to `1.72.24` also drops the `drbd.linbit.com` taint tolerations from `helm_lib_tolerations`, so pods rendered through it no longer tolerate `lost-quorum`, `force-io-error` and `ignore-fail-over`, and aligns the VPA `maxAllowed` memory to the recommender’s 64Mi quantum.
- **[istio]** Fixed CVEs in istio, kiali and operator images (golang.org/x/net, golang.org/x/text).
- **[multitenancy-manager]** Disabling the module now removes the cluster-objects grant webhooks, which used to stay behind and block writes in project namespaces.
  The grant webhooks are now removed together with the module. On a cluster where `multitenancy-manager` is already disabled at the update, the `cluster-objects-grants-defaulting` and `cluster-objects-grants-validator` webhook configurations left by earlier versions stay, and writes to grantable resources in project namespaces (for example, removing a PVC finalizer) keep failing with `service "multitenancy-manager" not found`. Delete them by hand: `d8 k delete mutatingwebhookconfiguration cluster-objects-grants-defaulting` `d8 k delete validatingwebhookconfiguration cluster-objects-grants-validator` Nothing has to be done on clusters where the module is enabled.
- **[multitenancy-manager]** The controller takes its leader lease with any number of replicas, so a rollout of a single-replica installation no longer has two pods upgrading project Helm releases at once.
  In single-replica installations a new controller pod reconciles projects only after the previous pod has stopped and released the lease. If the previous pod is lost without a shutdown, for example together with its node, the new pod starts reconciling projects 60 to about 78 seconds after it starts.
- **[multitenancy-manager]** The Project conversions of the next release are registered in advance, so reads and writes of projects keep working while webhook-handler is updated in HA mode.
- **[multitenancy-manager]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net`.
- **[multitenancy-manager]** The webhooks no longer leave zombie processes in webhook-handler until it runs out of pids.
- **[node-manager]** Fixed cluster-autoscaler using node group names without the cluster prefix after a Deckhouse restart, which stopped autoscaling and triggered the `D8ClusterAutoscalerTooManyErrors` alert.
- **[node-manager]** Fixed cluster-autoscaler stopping autoscaling of all node groups and restarting every two hours when an MCM node group has `minPerZone` equal to `maxPerZone`.
- **[node-manager]** Fixed CVEs in cluster-autoscaler Go dependencies (OpenTelemetry, and grpc and x/crypto for Kubernetes 1.31).
- **[node-manager]** A StaticInstance whose StaticMachine was deleted is returned to Pending and

becomes available for bootstrapping again instead of staying out of the pool forever.
- **[node-manager]** Fixed cluster-autoscaler not scaling CAPI node groups up from zero for pods requesting ephemeral storage, and putting the whole scale-up of a multi-zone node group into a single zone.
- **[node-manager]** Bump golang.org/x/crypto and OpenTelemetry in bashible-apiserver and capi-controller-manager.
- **[node-manager]** Fixed the CAPI MachineHealthCheck replacing cloud nodes that take longer than 5 minutes to become Ready after registering, and limited remediation to when at most half of the CAPI machines are unhealthy.
  The cloud MachineHealthCheck is updated in existing clusters. A node in Ready=False is replaced after 10 minutes instead of 5 (Ready=Unknown still after 5), and remediation stops while more than 50% of the cluster's CAPI machines are unhealthy.
- **[node-manager]** Fixed node-manager failing to apply its values in the BE, SE, SE+, EE, FE and CSE editions when a CAPI node group has a known boot disk size.
- **[registry]** Build docker-auth from the new fork repository at flant-1.0.0, which fixes CVE-2026-81870.
- **[registrypackages]** Fixed CVEs in crictl 1.31.
- **[service-with-healthchecks]** Go deps are updated.
  The controller and agents will be restarted.
- **[terraform-manager]** Bump opentofu to 1.12.7 to fix CVE-2026-56855, CVE-2026-78662.
- **[tools]** Remove d8-gpu namespace from integrity check.
- **[user-authn]** The `idTokenTTL` description and the `D8UserAuthnIDTokenTTLTooLong` alert say that a value of 6 hours or more is replaced with `5h59m` by the next settings version, and what lowering it does to DexAuthenticator sessions and issued ID tokens.
- **[user-authn]** Restrict the Job that issues the `basic-auth-proxy` certificate to the read-only front-proxy CA files and cover it with a SecurityPolicyException.
- **[user-authn]** A DexClient or a DexAuthenticator in a namespace whose name YAML reads as a number or a boolean no longer breaks the user-authn release.
- **[user-authn]** A DexAuthenticator with a name longer than 45 characters no longer logs its users out on every run of the hook.
- **[user-authn]** The DexAuthenticator validating webhook no longer logs a jq error for an object without additionalApplications.
- **[user-authn]** Errors the webhook scripts of user-authn, user-authz and multitenancy-manager write to stderr are logged as warnings instead of info messages.
- **[user-authn]** Protected applications keep serving users during a short dex outage and during restarts.
- **[user-authn]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net`.
- **[user-authn]** The webhooks no longer leave zombie processes in webhook-handler until it runs out of pids.
- **[user-authz]** The authorization webhook no longer restarts while the API server is unreachable, which could deadlock the control plane on bootstrap or an API server restart with multitenancy enabled.
- **[user-authz]** a `kube-apiserver` health probes no longer fail when `authz-webhook` is unavailable.
- **[user-authz]** The identity-assign check treats an identity with every permission across the cluster, such as a subject of `cluster-admin` bound by a ClusterRoleBinding, as SuperAdmin. Such identities can write ClusterAuthorizationRules, Users and Groups that carry `SuperAdmin` or `cluster-admin`. A ClusterAuthorizationRule gives this right only without `namespaceSelector` and `limitNamespaces` and, with multi-tenancy on, only with `allowAccessToSystemNamespaces: true`, even if it grants `cluster-admin`.
- **[user-authz]** The RBACv2 roles d8:manage:* and d8:use:role:* keep the rights of the modules that have already switched to the capability labels of the next release while the upgrade to it is in progress.
- **[user-authz]** The authorization webhook keeps applying ClusterAuthorizationRules when a `limitNamespaces` entry is not a valid regular expression; the entry is left out, and its rule covers fewer namespaces.
  Before, such an entry stopped the webhook from reloading the rules, so subjects of rules added later, and every subject after a restart of the webhook, were not limited to their namespaces. Subjects of a rule whose only entry is invalid now get no namespace through the entries of that rule.
- **[user-authz]** The `D8UserAuthzLegacyRBACv2CustomRoleFound` alert lists every custom role that selects capabilities by the `rbac.deckhouse.io/kind: manage|use` label or by the aggregation label of a lineage the next role model does not collect, or that loses its RoleBindings in the namespaces of the modules, and the FAQ describes how to prepare such a role before the update without losing access.
- **[user-authz]** An AuthorizationRule in a namespace whose name YAML reads as a number, a boolean or null no longer breaks the module's bindings.
- **[user-authz]** Managers who assign roles by range can again change DexProviders, UserOperations, Users and Groups of identities that hold a ClusterAuthorizationRule.
- **[user-authz]** The authorization webhook answers the requests in flight when it restarts instead of denying them.

- **[user-authz]** Fixed CVE-2026-78659, CVE-2026-78660, CVE-2026-78663, CVE-2026-78669 and CVE-2026-97032 in `golang.org/x/net` and CVE-2026-56851 in `golang.org/x/text`.
- **[user-authz]** The webhooks no longer leave zombie processes in webhook-handler until it runs out of pids.

1.77.2

Know before update

- A StaticInstance whose connectivity check fails now keeps its reservation for the whole bootstrap window instead of returning to the pool on every failed attempt, and the bootstrap (20 min) and cleanup (10 min) timeouts, which previously never fired, are now reachable. If nothing ever ran on the host, the bootstrap timeout returns the instance to the pool after 20 minutes as before. If the host was already bootstrapped in part, the instance returns only through MachineHealthCheck remediation (nodeStartupTimeoutSeconds: 1200) plus the cleanup timeout, so it becomes available again roughly 20 + 10 minutes after the failure, and remediation reboots the host as part of the cleanup.
- A cluster still on 1.76 whose user-authn ModuleConfig sets idTokenTTL to 6h or more updates to this release without Deckhouse crash-looping, and stays on it until idTokenTTL is lowered below 6h. A cluster already crash-looping on 1.77.0 or 1.77.1 does not recover by itself, since the release controller runs in the pod that is down; set failurePolicy Ignore on the module-configs.deckhouse-webhook.deckhouse.io webhook of the deckhouse-webhook ValidatingWebhookConfiguration, lower idTokenTTL below 6h, and restart the deckhouse pod.
- Clusters bootstrapped before v1.55 were switched to the `Baseline` default PSS policy in 1.77, which denies privileged workloads in user namespaces. After the update, the default policy on these clusters becomes less strict: it is switched back to `Privileged`, and privileged pods in user namespaces pass admission again. To keep the `Baseline` policy, set `settings.podSecurityStandards.defaultPolicy: Baseline` explicitly in the `admission-policy-engine` ModuleConfig.
- Clusters whose `d8-system/install-data` ConfigMap holds an empty or invalid version and whose `podSecurityStandards.defaultPolicy` isn’t set were switched to the `Baseline` default PSS policy in 1.77. After the update, the default policy on these clusters becomes less strict: it is switched back to `Privileged`. To keep the `Baseline` policy, set `settings.podSecurityStandards.defaultPolicy: Baseline` explicitly in the `admission-policy-engine` ModuleConfig.
- Deckhouse 1.78 makes `Baseline` the default PSS policy for all clusters. On clusters bootstrapped with a Deckhouse version lower than v1.55 or with an unknown bootstrap version, the `PodSecurityStandardsDefaultPolicyNotSet` alert fires, and the update to 1.78 is blocked until `settings.podSecurityStandards.defaultPolicy` is set explicitly in the `admission-policy-engine` ModuleConfig. To keep the current behavior, set `defaultPolicy: Privileged`. To switch to `Baseline` or `Restricted`, make sure that the workloads in non-system namespaces comply with the chosen policy first.
- In Deckhouse CSE, a DexProvider that already has `enableBasicAuth: true` keeps it after the update, and any change to its `oidc` or `ldap` section is rejected until `enableBasicAuth` is set to `false`. The same holds for an LDAP provider that combines `insecureNoSSL` with `startTLS`, `insecureSkipVerify` or `rootCAData`. Set `enableBasicAuth: false` and resolve the LDAP TLS settings on such providers.
- The `d8-kube-proxy` pods will be restarted on all nodes.
- Two Go hooks registered in one file shared a derived name, so the Kubernetes-binding half lost its bindings and the `d8_cluster_objects_grant_violated` series was recomputed only by the periodic scan. The series itself was never missing; it could lag a policy change by up to two minutes.
- dhctl converge no longer deadlocks on master replacement or externalIPAddresses removal in Yandex Cloud; the terraform-manager provider release is patched to detach one-to-one NAT before deleting a reserved address.

Features

- **[admission-policy-engine]** Added the `PodSecurityStandardsDefaultPolicyNotSet` alert that fires when the `install-data` ConfigMap is missing and the default PSS policy must be set explicitly in the ModuleConfig.
- **[admission-policy-engine]** The `PodSecurityStandardsDefaultPolicyNotSet` alert now fires when `podSecurityStandards.defaultPolicy` isn’t set and the `Privileged` policy is derived from the bootstrap version. The update to Deckhouse 1.78 is blocked until the policy is set explicitly.
  Deckhouse 1.78 makes `Baseline` the default PSS policy for all clusters. On clusters bootstrapped with a Deckhouse version lower than v1.55 or with an unknown bootstrap version, the `PodSecurityStandardsDefaultPolicyNotSet` alert fires, and the update to 1.78 is blocked until `settings.podSecurityStandards.defaultPolicy` is set explicitly in the `admission-policy-engine` ModuleConfig. To keep the current behavior, set `defaultPolicy: Privileged`. To switch to `Baseline` or `Restricted`, make sure that the workloads in non-system namespaces comply with the chosen policy first.
- **[deckhouse-controller]** Add the `x-deckhouse-immutable`, `x-deckhouse-ui-group`, and `x-`

`deckhouse-enum-switch-settings` extensions to Application package settings schemas; the Application webhook rejects changes to `x-deckhouse-immutable` fields after the settings were applied.
- **[deckhouse-controller]** When an Application is deleted, Deckhouse also deletes the objects of the kinds listed in the package's `orphanResources` value that the application left in its namespace.
- **[docs]** Add user-authn-controller, user-api and basic-auth-proxy components to user-authn module architecture section.
- **[docs]** Rename DKP->DP and replace PNG->SVG architecture section
- **[docs]** Fix links to architecture diagrams
- **[docs]** Add alb module architecture section.
- **[docs]** Add managed services modules architecture documentation section.
- **[docs]** Add sdn module architecture section.

Fixes

- **[admission-policy-engine]** Restored the `Privileged` default PSS policy for clusters bootstrapped before v1.55 (without the `install-data` ConfigMap).
  Clusters bootstrapped before v1.55 were switched to the `Baseline` default PSS policy in 1.77, which denies privileged workloads in user namespaces. After the update, the default policy on these clusters becomes less strict: it is switched back to `Privileged`, and privileged pods in user namespaces pass admission again. To keep the `Baseline` policy, set `settings.podSecurityStandards.defaultPolicy: Baseline` explicitly in the `admission-policy-engine` ModuleConfig.
- **[admission-policy-engine]** Fixed CVEs in the `gatekeeper` and `ratify` images.
- **[admission-policy-engine]** Fixed the default Pod Security Standards (PSS) policy being switched to `Baseline` without notice when the bootstrap version in the `d8-system/install-data` ConfigMap is empty or invalid. Such clusters now get the `Privileged` policy.
  Clusters whose `d8-system/install-data` ConfigMap holds an empty or invalid version and whose `podSecurityStandards.defaultPolicy` isn't set were switched to the `Baseline` default PSS policy in 1.77. After the update, the default policy on these clusters becomes less strict: it is switched back to `Privileged`. To keep the `Baseline` policy, set `settings.podSecurityStandards.defaultPolicy: Baseline` explicitly in the `admission-policy-engine` ModuleConfig.
- **[candi]** remove d8-bcheck in cleanup_static_node
- **[candi]** containerd integrity errors are written to `/var/log/containerd/containerd-integrity.log` on RED OS 8.
- **[cert-manager]** Fixed CVEs in the cert-manager images.
- **[cert-manager]** Gateway API integration is now enabled automatically when the Gateway and HTTPRoute CRDs are installed.
- **[chrony]** Bump golang.org/x/crypto to v0.55.0 in chrony_exporter to fix CVE-2026-56854.
- **[ci]** update werf to v2.74.2-dk.1
- **[ci]** Add allowed alerts for D8Monitoring for e2e which uses 1.77+
- **[ci]** The documentation changes validation resolves symlinked related language files.
- **[ci]** Fix changelog-generator image tag in changelog-update job to use the shared CI_IMAGES_TAG
- **[ci]** add skip docs validation label
- **[cloud-provider-dvp]** Fixed the `D8CloudProviderDVPMigrationPending` alert that kept firing after the migration resources were applied.
- **[cloud-provider-dvp]** Fixes CVE vulnerabilities in cloud-provider-dvp images.
- **[cloud-provider-dvp]** Fixes cluster bootstrap getting stuck when `capdvp-controller-manager` and the validation webhook cannot reach the API server and DNS before the CNI is ready.
- **[cloud-provider-openstack]** Fixes cluster bootstrap getting stuck when `capo-controller-manager` cannot reach the API server and DNS before the CNI is ready.
- **[cloud-provider-vcd]** Batch VM cache refresh on cache miss
- **[cloud-provider-vcd]** Fixes cluster bootstrap getting stuck when `capcd-controller-manager` cannot reach the API server and DNS before the CNI is ready.
- **[cloud-provider-yandex]** fix "Address in use" failures when replacing nodes and when removing external IP addresses
  dhctl converge no longer deadlocks on master replacement or externalIPAddresses removal in Yandex Cloud; the terraform-manager provider release is patched to detach one-to-one NAT before deleting a reserved address.
- **[common]** Fix kubelet podSandbox order
- **[common]** fixed private build cni-migration-*
- **[common]** Fix CVEs in the `kube-rbac-proxy` image.
- **[common]** Fixed known vulnerabilities in the CoreDNS image used by the node-local-dns module.
- **[control-plane-manager]** A new etcd `maxDbSize` is applied without a `deckhouse` restart.
- **[control-plane-manager]** Fix kubectl exec through Kubernetes API published with Gateway API.
- **[deckhouse]** set releaseChannel in CSE
- **[deckhouse]** Fix setting Scaled before Installed.
- **[deckhouse]** Remove leftover symlink-backend entries when deploying packages with the erofs backend.
- **[deckhouse]** Fix random startup failures with 'waitid - no child processes' when restoring erofs modules.

- **[deckhouse]** Fix CVEs in deckhouse-cli.
- **[dhctl]** Restoring of handling api_server_url and api_server_token in dhctl
- **[dhctl]** Bump google.golang.org/grpc to v1.83.2 to fix CVE-2026-84304 and CVE-2026-84445.
- **[dhctl]** Fixed terraform-auto-converger, terraform-state-exporter and `dhctl converge` failing on cloud clusters whose `cloud-provider-<name>` ModuleConfig has no `spec.enabled` field. dhctl now treats `spec.enabled` as optional and a ModuleConfig without it as disabled.
- **[dhctl]** Fixed the terraform-auto-converger init container failing on clusters already migrated to OpenTofu - the migration check now skips planning when no Terraform state is stored in the cluster.
- **[docs]** Fixed FAQ section for ansible inventory.
- **[docs]** Register the ports of csi-hitachi and csi-mind-ustor, list the registry storage ports, and drop storage ports that are no longer bound in the host network from the ports reference.
- **[docs]** Fix search hanging when IndexedDB upgrade is blocked.
- **[docs]** Removed mentions of Cilium from the inter-cluster cooperation documentation page.
- **[istio]** Fixed the istio module getting stuck when switching `globalVersion` from 1.21 while proxies are still connected to the 1.21 control plane.
- **[istio]** Fix leaking istiod control plane after removing an Istio version or disabling the module.
- **[istio]** Switching back to Istio 1.21 no longer removes the `networking.istio.io/v1` API from the Istio CRDs and no longer breaks the istio release.
- **[istio]** Deckhouse update requirements now take into account Istio control planes that are still being removed.
- **[istio]** The Istio operator now recovers Helm releases stuck in the `pending-*` or `uninstalling` state.
- **[kube-dns]** Set resource limits for the `render-etc-hosts-with-cluster-domain-aliases` init container so that pods can be created in namespaces with a ResourceQuota on `limits.cpu/limits.memory`.
- **[kube-proxy]** Fixed kube-proxy failing to start on Kubernetes 1.36 and higher because of the removed `TopologyAwareHints` feature gate.
  The `d8-kube-proxy` pods will be restarted on all nodes.
- **[metallb]** Fixed creation of `BGPPeer` objects on Kubernetes 1.36+, which blocked BGP sessions and LoadBalancer IP announcements in BGP mode.
- **[multitenancy-manager]** The grant-violation hook re-evaluates policies on change again, not only on its two-minute schedule.
  Two Go hooks registered in one file shared a derived name, so the Kubernetes-binding half lost its bindings and the `d8_cluster_objects_grant_violated` series was recomputed only by the periodic scan. The series itself was never missing; it could lag a policy change by up to two minutes.
- **[multitenancy-manager]** Fixed vulnerabilities in the `multitenancy-manager` image.
- **[node-manager]** Increase the capi-controller-manager VPA maximum memory recommendation to prevent health-check failures and restarts under memory pressure.
- **[node-manager]** Stop caps-controller-manager from rewriting StaticInstance objects in a hot loop when a connectivity check fails, which caused a sustained load on etcd.
  A StaticInstance whose connectivity check fails now keeps its reservation for the whole bootstrap window instead of returning to the pool on every failed attempt, and the bootstrap (20 min) and cleanup (10 min) timeouts, which previously never fired, are now reachable. If nothing ever ran on the host, the bootstrap timeout returns the instance to the pool after 20 minutes as before. If the host was already bootstrapped in part, the instance returns only through MachineHealthCheck remediation (nodeStartupTimeoutSeconds: 1200) plus the cleanup timeout, so it becomes available again roughly 20 + 10 minutes after the failure, and remediation reboots the host as part of the cleanup.
- **[node-manager]** Fix duplicate NVIDIA GPU dashboards when the gpu module is enabled.
- **[node-manager]** Bump google.golang.org/grpc to v1.83.2 in bashible-apiserver to fix CVE-2026-84304 and CVE-2026-84445.
- **[node-manager]** Bump golang.org/x/crypto to v0.55.0 and google.golang.org/grpc to v1.83.2 in capi-controller-manager to fix CVE-2026-56854, CVE-2026-84304 and CVE-2026-84445.
- **[node-manager]** Fixes CVE vulnerabilities in caps-controller-manager.
- **[node-manager]** Hooks no longer fail the module run when a NodeGroup, Node or NodeUser is deleted between the snapshot and the patch. Such failures delayed the readiness of the `deckhouse` pod in clusters that delete NodeGroups often.
- **[registry]** Bump dependencies in registry.
- **[registry]** Bump golang.org/x/crypto in nodeservices-manager to fix CVE-2026-56854, add VEX statements for x/crypto/ssh findings.
- **[registrypackages]** Bumped `containerd` to 1.7.36 / 2.2.9 and OpenTelemetry to 1.45.0 in `containerd` and `crictl` to fix CVEs.
- **[service-with-healthchecks]** Fixed metrics endpoints being reachable without going through kube-rbac-proxy authorization.
- **[terraform-manager]** Bump google.golang.org/grpc to v1.82.2 in opentofu to fix CVE-2026-84445. CVE-2026-84304 is not exploitable in opentofu and is covered by VEX.
- **[terraform-manager]** Fixed the `to-tofu-migrator` init container of `terraform-auto-converger` failing to resolve the provider bundle digest, which kept auto-converge from starting.
- **[user-authn]** An idTokenTTL of 6h or more no longer stops Deckhouse from starting; it raises the D8UserAuthnIDTokenTTLTooLong alert and blocks the update to the next minor release until it is

lowered.
A cluster still on 1.76 whose user-authn ModuleConfig sets idTokenTTL to 6h or more updates to this release without Deckhouse crash-looping, and stays on it until idTokenTTL is lowered below 6h. A cluster already crash-looping on 1.77.0 or 1.77.1 does not recover by itself, since the release controller runs in the pod that is down; set failurePolicy Ignore on the module-configs.deckhouse-webhook.deckhouse.io webhook of the deckhouse-webhook ValidatingWebhookConfiguration, lower idTokenTTL below 6h, and restart the deckhouse pod.
- **[user-authn]** Fixed CVEs in the module images.
- **[user-authn]** In Deckhouse CSE, a DexProvider with `enableBasicAuth` is rejected, since the edition does not ship the basic authentication proxy; the CSE copy of the CRD also gets the LDAP TLS checks of the module.
  In Deckhouse CSE, a DexProvider that already has `enableBasicAuth: true` keeps it after the update, and any change to its `oidc` or `ldap` section is rejected until `enableBasicAuth` is set to `false`. The same holds for an LDAP provider that combines `insecureNoSSL` with `startTLS`, `insecureSkipVerify` or `rootCAData`. Set `enableBasicAuth: false` and resolve the LDAP TLS settings on such providers.
- **[user-authz]** Fixed CVEs in the `permission-browser-apiserver` image.
- **[user-authz]** The authorization webhook answers 500 and logs the real error instead of terminating when it cannot marshal a response.

1.77.1

Know before update

- A Deckhouse release suspended on its release channel is no longer applied by clusters that are behind and reach it through a step-by-step update. The suspend flag lives only in the release-channel image; previously it was dropped when the target release was built from its per-version image, so lagging clusters updated to a suspended release anyway.
- After the update, the controller adds the annotations and labels of every ServiceWithHealthchecks to the Service created for it. If a ServiceWithHealthchecks of the `LoadBalancer` type carries annotations that configure the load balancer (`network.deckhouse.io/load-balancer-ips`, `network.deckhouse.io/load-balancer-shared-ip-key` and similar), the load balancer controller applies them and may assign a different address to the service, which causes a short interruption of the connections.
- CAPI NodeGroups whose MachineDeployment was stuck on an old MachineTemplate are rolled to the current InstanceClass after the update, within maxSurgePerZone/maxUnavailablePerZone. Groups already on the current template are not touched.
- D8UserAuthzRenderSlow requires at least three renders averaging above 30 seconds over 30 minutes, D8UserAuthzReleaseSlow at least two releases averaging above 5 minutes over one hour, and both must hold for 15 minutes. Severities are one step lower (6 and 7). Their descriptions now point at CPU shortage on the node rather than at the number of ClusterAuthorizationRules, which does not affect render time.
- During the first `user-authz` release after the update the per-role bindings are protected from the release engine, the aggregated ones are created, and the per-role bindings are deleted right after the release. Permissions granted through annotated ClusterRoles stay in place throughout. The `user-authz.deckhouse.io/access-level` label is now set automatically on annotated ClusterRoles. In audit logs, access granted through annotated ClusterRoles is attributed to `user-authz:<level>:custom` instead of the individual ClusterRole.
- Endpoints for pods in a terminal phase (Failed/Succeeded) are no longer published. In DVP clusters this prevents traffic from being routed to a VirtualMachine IP that has been reused by another pod. Pods being deleted are now published with the serving and terminating conditions, which enables the graceful shutdown flow for consumers. Pod readiness is derived from the PodReady condition, and stale probe results are reset when a pod becomes not ready, is recreated, or changes its IP.
- The `descheduler` Deployment in `d8-descheduler` restarts once on upgrade to pick up the new argument. Descheduling behavior is unchanged for workloads that do not implement their own eviction policy.
- The `service-with-healthchecks` controller and agent are restarted. A `ServiceWithHealthchecks` that was left in `NotAllEndpointsAreReady` with no EndpointSlice recovers automatically once the updated agent starts, no manual action is needed.
- The installer, the deckhouse ServiceAccount, holders of the cluster administrator kubeconfig and the kube-system / d8-system ServiceAccounts can create a User or a Group named by an existing authorization rule without the acknowledgement annotation. Every other requester is checked as before.

Features

- **[candi]** Base images updated to v3
- **[descheduler]** Always run the descheduler with the upstream `EvictionsInBackground` feature gate enabled.
  The `descheduler` Deployment in `d8-descheduler` restarts once on upgrade to pick up the new argument. Descheduling behavior is unchanged for workloads that do not implement their own eviction policy.
- **[docs]** Add managed-valkey module architecture section.

- **[docs]** Refactor build pipeline and add CSE edition support.
- **[docs]** Update virtualization module architecture section according to release v1.10.0
- **[docs]** Add monitoring-kubernetes module architecture section.
- **[docs]** Add documentation for Managed PostgreSQL service.
- **[docs]** Add extended-monitoring module architecture section.
- **[docs]** Add registry-packages-proxy module architecture section.
- **[docs]** Expand the documentation for virtualization.
- **[docs]** Add istio module architecture section.
- **[registry]** Migration part of new registry implementation.
- **[user-authn]** Update the Dex authentication page to the new Deckhouse branding, with light and dark themes selected by the user's system settings.

Fixes

- **[candi]** Fix hanging sysctl tuner on nodes with many loop and dm devices
- **[candi]** containerd v2 no longer starts without the `erofs` snapshotter, and loads the `erofs` module before start.
- **[cert-manager]** Restore the `cert-manager` and `cainjector` permissions that were lost against the upstream chart, including access to `ListenerSet` resources.
- **[chrony]** Fixed the unit-detection condition in the `disable-ntp-on-node.sh` node step.
- **[cloud-provider-aws]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-aws]** Fixed CVEs in the cloud-provider-aws module.
- **[cloud-provider-azure]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-azure]** Fix multimaster bootstrap by falling back to the `d8-masters-kubernetes-data-device-path` Secret when the master data device LUN is not provided via cloud-init.
- **[cloud-provider-dvp]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-dvp]** Fixed CVEs in the cloud-provider-dvp module.
- **[cloud-provider-gcp]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-gcp]** Fixed CVEs in the cloud-provider-gcp module.
- **[cloud-provider-huaweicloud]** Fixed CVEs in the cloud-provider-huaweicloud module.
- **[cloud-provider-openstack]** Fixed CVEs in the cloud-provider-openstack module.
- **[cloud-provider-yandex]** Fixed CVEs in the cloud-provider-yandex module.
- **[cloud-provider-zvirt]** Resize the boot disk of a recreated VM, let a replaced VM actually power off, and recreate the VM when customNetworkConfig changes.
- **[deckhouse]** Exclude all Python test files from webhook image.
- **[deckhouse]** Keep `x-kubernetes-sensitive-data` on CRDs that have a conversion webhook.
- **[deckhouse]** Allow editing and deleting the resources of an application under maintenance.
- **[deckhouse-controller]** Honor a channel-level release suspend for clusters that reach the suspended version through a step-by-step update.
  A Deckhouse release suspended on its release channel is no longer applied by clusters that are behind and reach it through a step-by-step update. The suspend flag lives only in the release-channel image; previously it was dropped when the target release was built from its per-version image, so lagging clusters updated to a suspended release anyway.
- **[deckhouse-controller]** Stop the module source registry fan-out from replaying and flooding the main queue with duplicate moduleRun tasks.
- **[deckhouse-controller]** Do not commit the package repository registry checksum when no application could be annotated, which left them all on stale registry settings.
- **[docs]** Fix broken JUnit XML report links on the Kubernetes conformance page.
- **[docs]** Clarify global storageClass behavior for Prometheus
- **[docs]** Fix alert rendering in OpenAPI specs.
- **[istio]** Waypoints are no longer advertised as cross-network east-west gateways.
- **[istio]** Enable Istio Gateway API deployment and gatewayclass controllers for istiod 1.27 and 1.29.
- **[istio]** Document that ambient-mode workloads cannot take part in federation or multicluster.
- **[istio]** Fix istiod RBAC rules that drifted away from upstream.
- **[multitenancy-manager]** Refresh grant webhook CA bundles after the admission certificate is rotated.
- **[node-manager]** node-controller reads the CAPI instance-class checksum from a helm-rendered ConfigMap instead of a possibly stale MachineTemplate, so MachineDeployments follow InstanceClass changes and their bootstrap Secrets keep a valid token.
  CAPI NodeGroups whose MachineDeployment was stuck on an old MachineTemplate are rolled to the current InstanceClass after the update, within maxSurgePerZone/maxUnavailablePerZone. Groups already on the current template are not touched.
- **[node-manager]** node-controller no longer removes taints set by other components on the first reconcile of Static and CloudPermanent nodes.
- **[registry]** Fix the order in which the Unmanaged params are stored for the fallback to the Legacy mode.
- **[registry]** Fix the security context in the registry module for CSE.
- **[registry-packages-proxy]** Bump golang.org/x/{crypto,net,sys,text} to clear 24 CVEs, including CVE-2026-56854 (CRITICAL) and CVE-2026-25681, CVE-2026-39821, CVE-2026-46600 (HIGH).
- **[registrypackages]** Fix dm-verity devices that could never be closed for containerd 2.2.7 (CSE only).
- **[registrypackages]** Bumped containerd to 1.7.35/2.2.8; bumped grpc, x/crypto, and websocket

in containerd, crictl, and Kubernetes 1.32–1.36 gomod patches.
- **[registrypackages]** Fsync the EROFS layer blob in `containerd` before committing snapshot metadata, so an unclean shutdown can no longer leave an unmountable image layer.
- **[service-with-healthchecks]** Annotations and labels of a ServiceWithHealthchecks are now copied to the Service created for it.
  After the update, the controller adds the annotations and labels of every ServiceWithHealthchecks to the Service created for it. If a ServiceWithHealthchecks of the `LoadBalancer` type carries annotations that configure the load balancer (`network.deckhouse.io/load-balancer-ips`, `network.deckhouse.io/load-balancer-shared-ip-key` and similar), the load balancer controller applies them and may assign a different address to the service, which causes a short interruption of the connections.
- **[service-with-healthchecks]** Stopped publishing terminated pods in EndpointSlices and started publishing pods being deleted as terminating endpoints.
  Endpoints for pods in a terminal phase (Failed/Succeeded) are no longer published. In DVP clusters this prevents traffic from being routed to a VirtualMachine IP that has been reused by another pod. Pods being deleted are now published with the serving and terminating conditions, which enables the graceful shutdown flow for consumers. Pod readiness is derived from the PodReady condition, and stale probe results are reset when a pod becomes not ready, is recreated, or changes its IP.
- **[service-with-healthchecks]** Fixed a load balancer losing all its endpoints for good after a target pod was replaced.
  The `service-with-healthchecks` controller and agent are restarted. A `ServiceWithHealthchecks` that was left in `NotAllEndpointsAreReady` with no EndpointSlice recovers automatically once the updated agent starts, no manual action is needed.
- **[tools]** Bump d8-cli to v0.33.17 - `d8 mirror push` restores the release channel tags of the installer image, so `docker run <repo>/install:<channel>` works against a mirrored registry
- **[tools]** Bump d8-cli to v0.33.19 - closes the CVE findings in the bundled d8 binary (grpc 1.83.2 and x/image 0.45.0) and adds resume of interrupted d8 data and snapshot transfers
- **[user-authz]** The identity collision check no longer applies to the platform's own identities, so a retried bootstrap creates the initial User again instead of timing out.
  The installer, the deckhouse ServiceAccount, holders of the cluster administrator kubeconfig and the kube-system / d8-system ServiceAccounts can create a User or a Group named by an existing authorization rule without the acknowledgement annotation. Every other requester is checked as before.
- **[user-authz]** Grant custom ClusterRoles (annotated with `user-authz.deckhouse.io/access-level`) through one aggregated ClusterRole per access level, so the number of bindings per ClusterAuthorizationRule/AuthorizationRule no longer depends on the number of such roles. During the first `user-authz` release after the update the per-role bindings are protected from the release engine, the aggregated ones are created, and the per-role bindings are deleted right after the release. Permissions granted through annotated ClusterRoles stay in place throughout. The `user-authz.deckhouse.io/access-level` label is now set automatically on annotated ClusterRoles. In audit logs, access granted through annotated ClusterRoles is attributed to `user-authz:<level>:custom` instead of the individual ClusterRole.
- **[user-authz]** Group the module alerts under a group named apart from the alerts themselves, so the incident shelf accepts them.
- **[user-authz]** The D8UserAuthzRenderSlow and D8UserAuthzReleaseSlow alerts no longer fire on a single slow render or release, such as a cluster bootstrap under CPU pressure or the one-time migration pass.
  D8UserAuthzRenderSlow requires at least three renders averaging above 30 seconds over 30 minutes, D8UserAuthzReleaseSlow at least two releases averaging above 5 minutes over one hour, and both must hold for 15 minutes. Severities are one step lower (6 and 7). Their descriptions now point at CPU shortage on the node rather than at the number of ClusterAuthorizationRules, which does not affect render time.

Version 1.76

Important

- The fencing-agent component of the `node-manager` module now uses a gossip-based protocol (memberlist library) for distributed node health monitoring. This reduces the risk of false worker node reboots when the control plane is unavailable or the API server is under heavy load.

- Fixed an issue where `cluster-autoscaler` could get stuck after a failed machine creation in the cloud (for errors other than `ResourceExhausted`). The cluster now automatically recovers its ability to scale after temporary cloud provider failures.

Deckhouse subsystem

- Added support for deploying a DP cluster using the `Proxy` mode for accessing the container registry.

- `Kubectl` on cluster nodes has been replaced with the `d8 k` alias (Deckhouse CLI utility).

- Changes in `dhctl` (CLI installer):
  - Added support for running `dhctl` as a standalone binary without downloading the full image with all dependencies. Required components are pulled from the registry during execution, reducing startup time.
  - Added validation of public SSH keys. Invalid keys are now detected earlier, before they can cause connection issues.
  - Improved handling of TLS certificates for registries. The system certificate pool is now used together with a custom CA.
  - Added validation of InstanceClass against the selected cloud provider before installation, allowing early detection and installation abort in case of misconfiguration.
  - Fixed namespace configuration during bootstrap. `Dhctl` now updates settings even if the namespace already exists.
- Improved multi-master resiliency. Kubernetes versions are now correctly detected even when some master nodes are unavailable. Kube-apiserver no longer routes requests to etcd nodes in the learner state.

Kubernetes & Scheduling subsystem

- The `d8-cluster-kubernetes` ConfigMap now includes supported Kubernetes versions (`supportedVersions`), versions available for upgrade and downgrade (`availableVersions`), the current automatic version (`automaticVersion`).
- Changes in the `descheduler` module:
  - Upgraded to version 0.35.1. Added namespace filtering via label selectors and other improvements (see the module changelog for details).
  - Added the `deschedulingInterval` parameter to control how often descheduling runs: `Frequent`, `Moderate`, or `Rare`.
  - Added the `RemovePodsHavingTooManyRestarts` strategy, which evicts pods exceeding a restart threshold, freeing node resources from pods in `CrashLoopBackOff` state.
  - The module now automatically uses the Kubernetes Metrics API when the `metrics.k8s.io` API group is available in the cluster. This provides real resource consumption data for strategies to use.
- The `vertical-pod-autoscaler` module updated to version 1.6.0. Key changes:
  - The `InPlaceOrRecreate` mode is now used by default. In this mode, VPA attempts to adjust pod requests and limits without recreating pods (in-place updates).
  - Reduced likelihood of pod restarts when in-place updates are not possible (via the `--in-place-skip-disruption-budget` flag), also `minReplicas` value is no longer considered for in-place updates.
- VPA memory recommendations are now rounded to 64 MiB, improving readability and reducing unnecessary pod restarts caused by minor recommendation fluctuations.
- For Kubernetes 1.34 and later, the `DRAExtendedResource` feature gate is enabled by default.
- Added the `D8EtcdHighRequestsLatency` alert to detect high etcd request latency, which can lead to API server timeouts. A corresponding metric has been added to the “Control Plane Status” dashboard.

IAM subsystem

- Changes in the `user-authn` module:
  - Disabled the insecure OAuth 2.0 Implicit Flow. In that flow, access tokens are passed in URL fragments, posing leakage risks (for example, via the `Referer` header, browser history, or logs). Any integrations using the Authorization Code Flow are unaffected.
  - Added SAML 2.0 support via the DexProvider resource. DP clusters can now be connected to such identity providers as AD FS, Okta, Keycloak, OneLogin, and Shibboleth.
  - Added refresh token support for SAML in Dex. This allows DexAuthenticator и kubeconfig-generator to refresh tokens without a manual user login.
  - Added Single Logout (SLO) support. The identity provider can end user sessions via

`/saml/slo/{connector}`.

- Fixed stale cache issue on the login page by adding a Dex build-based `cache` parameter to the CSS URL.

- In multitenancy mode, users without permissions defined via ClusterAuthorizationRule no longer see all namespaces in the Deckhouse web UI. The default behavior is now deny-by-default: users get an empty namespace list unless access is explicitly granted. This does not affect privileged groups (`system:masters`, `kubeadm:cluster-admins`). Documentation of the `user-authz` module now includes an example for granting access to all namespaces.
- Added support for the `service-with-health-checks` module in isolated projects. A NetworkPolicy exception for the `d8-service-with-healthchecks` namespace has been added to the project template, which allows module agents to verify service availability and correctly operate under conditions of network isolation.

Security subsystem

- Gatekeeper upgraded to 3.22.0 and Ratify to 1.4.0. Key changes include a unified CEL and Rego policy behavior, added namespace context support in policies, and a CLI tool for policy benchmarking.
- The `denyVulnerableImages` setting has been moved from `admission-policy-engine` to `operator-trivy`, making it the single owner of this logic.
- The `cert-manager` module upgraded to 1.20.0. Key changes include added support for Azure DNS Private Zones in the DNS01 challenge, added annotation `acme.cert-manager.io/http01-ingress-ingressclassname` for redefining IngressClasses in HTTP-01 solvers, and security fixes.
- Updated vulnerable Go dependencies in `multitenancy-manager` (`go-jose`, `golang.org/x/crypto`, `grpc`, etc.).

Cluster & Infrastructure subsystem

- Changes in the `node-manager` module:
  - Improved draining hook event generation for better observability of the node eviction process.
  - Fixed StaticInstance issue causing infinite infrastructure deletion loops. That prevents infinite cleanup and queueing cycles.
  - Apiserver-proxy rewritten from NGINX to native Go implementation for better stability and reconnection speed.
- Changes in the `cloud-provider-dvp` module:
  - Added hybrid cluster support. Static master nodes and cloud worker nodes can now be used in a single cluster.
  - Improved converge reliability. Reworked the mechanism of resource readiness check and added the fail-fast behavior for possible errors when creating the infrastructure.
  - Added default StorageClass inheritance for Cluster API child clusters.
  - Improved VM lifecycle handling, reducing potential problems when deleting VMs and associated objects.
  - Added support for the `masterNodeGroup.instanceClass.additionalDisks` parameter that lets you configure additional disks for master nodes.
- Changes in the `cloud-provider-aws` module:
  - Spot node draining logic was moved into DP, making the spot node handling more unified.
  - Fixed handling of regions without `DescribeInstanceTopology`, reducing the number of false IAM errors.
- Changes in the `cloud-provider-openstack` module:
  - Added `ipMode: Proxy` support for LoadBalancer in Kubernetes 1.32 and later.
  - Added the `csiDriver.fsGroupPolicy` parameter for the CSI driver, which defines whether the volume supports changing owner and permissions before mounting.
- Changes in the `cloud-provider-vsphere` module:

    - Enabled the CSI snapshotter, enabling volume snapshotting.
    - Fixed zone and datastore detection. Now the only considered zones are the ones specified in the provider configuration.
- In the `cloud-provider-azure` module, added support for NVMe disks on Gen2 VMs.
- In the `cloud-provider-gcp` module, added `enableNestedVirtualization` and `additionalDisks` parameters to GCPInstanceClass, which let you configure nodes for workloads using virtualization and storage scenarios.
- In the `cloud-provider-yandex` module, fixed converge failure when removing externalIPAddresses.
- DVP, Huawei Cloud, and zVirt providers were migrated to Cluster API `v1beta2`.
- For Huawei Cloud and VCD providers, added SecurityPolicyException resources and enabled security policy validations.

Network subsystem

- Added support for handling incoming traffic via Gateway API using the `alb` module.
- Changes in the `cni-cilium` module:
    - Added conntrack table synchronization for live VM migration.
    - Added ICMP (ping) support via ExternalIP for LoadBalancer services with MetalLB.
- Added support for switching between supported CNI plugins in DP clusters.
- In the `node-local-dns` module, added `disableIPv6` parameter to disable IPv6 DNS resolution.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.76.18

Know before update

- All ingress-nginx controllers' pods will be restarted.
- Cilium agent pods will be restarted.
- Cilium agent pods will be restarted. With DSR, the DSR IPv4 option is stripped before the packet reaches the pod.
- Flannel pods will be restarted.
- Flannel pods will be restarted. The guard metrics use host port 4294.
- In ambient mode with Istio 1.27 or later, on nodes with legacy iptables, `istio-cni-node` switches from nftables to legacy iptables. Pods that were added to the mesh before the update keep their nftables rules. Restart these pods after the update. Otherwise, removing such a pod from the mesh without a restart leaves its old redirect rules in place, and the pod loses network connectivity.
- Pods using the updated nginx images roll out in the affected modules, including the ingress failover proxy where enabled.
- TLS connections from nodes to hosts whose certificates chain to `emSign ECC Root CA - G3` or Comodo `AAA Certificate Services` (eMudhra / emSign issued) will fail. Reissue such certificates from another CA before the update.
- The Istio pods will be restarted. Application sidecars will be updated only after the application pods are recreated.
- loki
- simple-bridge pods will be restarted.
- simple-bridge pods will be restarted. The guard metrics use host port 4294.

Features

- **[cni-cilium]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new clusters, `Audit` for existing ones).
  Cilium agent pods will be restarted. With DSR, the DSR IPv4 option is stripped before the packet reaches the pod.
- **[cni-flannel]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new clusters, `Audit` for existing ones).
  Flannel pods will be restarted. The guard metrics use host port 4294.
- **[cni-simple-bridge]** Added a CVE-2026-98432 mitigation that filters IPv4 TCP SYN packets with IP options in the host and pod network namespaces (`ipv4OptionsGuard`; `Enforce` for new

clusters, `Audit` for existing ones).
simple-bridge pods will be restarted. The guard metrics use host port 4294.

Fixes

- **[candi]** Bump base images to v3.1.28.
- **[cloud-provider-vcd]** Reduce node initialization delays and restore automatic cleanup of orphaned Nodes in VCD clusters.
- **[cni-cilium]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so Cilium agents start on Deckhouse Engine nodes.
  Cilium agent pods will be restarted.
- **[cni-flannel]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so Flannel pods start on Deckhouse Engine nodes.
  Flannel pods will be restarted.
- **[cni-simple-bridge]** The ipv4-options-guard sidecar no longer writes to the host `/etc`, so simple-bridge pods start on Deckhouse Engine nodes.
  simple-bridge pods will be restarted.
- **[common]** Updated the shared static nginx binary to 1.30.7 to address security vulnerabilities. Pods using the updated nginx images roll out in the affected modules, including the ingress failover proxy where enabled.
- **[deckhouse]** Show the Terminating phase while package repositories are being deleted
- **[deckhouse]** ModuleSource scans skip modules whose release channel has not changed, using a manifest HEAD request instead of pulling the release image.
- **[deckhouse-controller]** Module sources now scan the registry every 30 minutes by default, reducing registry load
- **[deckhouse-controller]** Prevent module hooks from running with unavailable snapshots while waiting for Deckhouse to restart after a module update
- **[deckhouse-controller]** A pending module release no longer loses its Kubernetes, Deckhouse and module requirements, so an update that the cluster does not satisfy stays pending instead of disabling the module.
- **[deckhouse-controller]** Pre-download modules enabled by the bundle when their ModuleConfig omits enabled, preventing upgrades from getting stuck.
- **[deckhouse-controller]** Releases of a still-embedded module no longer raise manual approval alerts once the module has a deployed release.
- **[dhctl]** Added the ability to retry a timed-out node drain during converge, and raised drain attempts from 5 to 10.
- **[docs]** Added registry module port 5001 to Deckhouse ports documentation
- **[docs]** Update terminology in the documentation and SEO improvements.
- **[go_lib]** Package proxy streams the full package body when the cache skips writing it.
- **[ingress-nginx]** Nginx was updated to 1.30.7.
  All ingress-nginx controllers' pods will be restarted.
- **[istio]** Fixed CVEs in istio images (x/net, x/crypto, klauspost/compress, x/text).
  The Istio pods will be restarted. Application sidecars will be updated only after the application pods are recreated.
- **[istio]** Fixed ambient mode on nodes with legacy iptables, where `istio-cni-node` crashed on Istio 1.25 and used nftables instead of legacy iptables on Istio 1.27 and later.
  In ambient mode with Istio 1.27 or later, on nodes with legacy iptables, `istio-cni-node` switches from nftables to legacy iptables. Pods that were added to the mesh before the update keep their nftables rules. Restart these pods after the update. Otherwise, removing such a pod from the mesh without a restart leaves its old redirect rules in place, and the pod loses network connectivity.
- **[loki]** fix kind version grafanaDatasource. Disable module loki block queue without module prometheus
  loki
- **[node-manager]** kube-rbac-proxy in capi-controller-manager no longer listens on all addresses of a node that has no addresses yet, so it does not take the controller's port.
- **[registry-packages-proxy]** Fixed registry-packages-proxy hanging after a cache error and serving corrupted packages, which blocked node updates.
- **[registrypackages]** Nodes stop trusting `emSign ECC Root CA - G3` and Comodo `AAA Certificate Services`, the roots behind the compromised eMudhra intermediate CAs.
  TLS connections from nodes to hosts whose certificates chain to `emSign ECC Root CA - G3` or Comodo `AAA Certificate Services` (eMudhra / emSign issued) will fail. Reissue such certificates from another CA before the update.

1.76.17

Know before update

- A diff will appear in the Terraform state for the base infrastructure: the E zone will be added. Because of this, the Terraform check will fail on the first master node. Run `dhctl converge` for the base infrastructure, and everything should get back to normal.

Features

- **[cloud-provider-yandex]** Add support for the `ru-central1-e` zone in Yandex Cloud.
  A diff will appear in the Terraform state for the base infrastructure: the E zone will be added. Because of this, the Terraform check will fail on the first master node. Run `dhctl converge` for the base infrastructure, and everything should get back to normal.
- **[documentation]** The in-cluster documentation now includes the guides.

Fixes

- **[candi]** Restored the lost `kubelet-disable-k-panic-check` patch for k8s 1.35, so kubelet no longer enforces `kernel.panic=10` on this version.
- **[ci]** add skip docs validation label
- **[ci]** Delete tests registry images
- **[cloud-provider-azure]** use .normal.apiserverEndpoints in data device Secret fallback
- **[cloud-provider-vsphere]** Fixed master node creation when the datastore is set as a path with parent folders.
- **[deckhouse]** Restore kubectl plugin lookup in d8 k and d8 kubectl
- **[dhctl]** Prevent dhctl converge from panicking when kubeconfig is used without an SSH host.
- **[docs]** Removed mentions of Cilium from the inter-cluster cooperation documentation page.
- **[docs]** Added experimental module enable note to the lifecycle badge.
- **[docs]** Rendered all versions from oss.yaml versions array.
- **[istio]** The Istio operator now recovers Helm releases stuck in the `pending-*` or `uninstalling` state.

1.76.16

Features

- **[docs]** Add managed services modules architecture documentation section.

Fixes

- **[ci]** Fix changelog-generator image tag in changelog-update job to use the shared CI_IMAGES_TAG
- **[cloud-provider-vcd]** Fixes cluster bootstrap getting stuck when `capcd-controller-manager` cannot reach the API server and DNS before the CNI is ready.
- **[istio]** Aligned CNI templates with upstream and fixed Istio 1.25 compatibility.
- **[istio]** Fixed the istio module getting stuck when switching `globalVersion` from 1.21 while proxies are still connected to the 1.21 control plane.
- **[istio]** Fix leaking istiod control plane after removing an Istio version or disabling the module.
- **[istio]** Switching back to Istio 1.21 no longer removes the `networking.istio.io/v1` API from the Istio CRDs and no longer breaks the istio release.

1.76.15

Know before update

- A StaticInstance whose connectivity check fails now keeps its reservation for the whole bootstrap window instead of returning to the pool on every failed attempt, and the bootstrap (20 min) and cleanup (10 min) timeouts, which previously never fired, are now reachable. If nothing ever ran on the host, the bootstrap timeout returns the instance to the pool after 20 minutes as before. If the host was already bootstrapped in part, the instance returns only through MachineHealthCheck remediation (nodeStartupTimeoutSeconds: 1200) plus the cleanup timeout, so it becomes available again roughly 20 + 10 minutes after the failure, and remediation reboots the host as part of the cleanup.
- The `self-signed-generator` init container now carries the fixed `10m` / `25Mi` it was always meant to have, the same as the identical init container of `user-api`, instead of the sum of the main container limits. A DexAuthenticator that set large limits no longer gets that sum as burst: generating the self-signed certificate then takes 3 to 4 seconds instead of under a second. It happens once per pod start and needs no action.
- log-shipper

Features

- **[docs]** Add istio module architecture section.
- **[docs]** Add user-authn-controller, user-api and basic-auth-proxy components to user-authn module architecture section.

Fixes

- **[cert-manager]** Restore the `cert-manager` and `cainjector` permissions that were lost against the upstream chart, including access to `ListenerSet` resources.
- **[chrony]** Fixed the unit-detection condition in the `disable-ntp-on-node.sh` node step.
- **[cloud-provider-aws]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-azure]** Fix multimaster bootstrap by falling back to the `d8-masters-kubernetes-data-device-path` Secret when the master data device LUN is not provided via

cloud-init.
- **[cloud-provider-azure]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-dvp]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-gcp]** fix GetCapacity not implemented error spam in logs
- **[cloud-provider-vcd]** Batch VM cache refresh on cache miss
- **[cloud-provider-yandex]** fix “Address in use” failures when replacing nodes and when removing external IP addresses
- **[deckhouse]** In upmeter, the `monitoring-and-autoscaling/alertmanager` probe is disabled by default and is turned on with the new `alertmanagerProbe.enabled` setting.
- **[deckhouse]** Fixed the CVEs reported for the v1.76.14 scan slice across all Observability team module images by bumping grpc to v1.83.2, the OpenTelemetry family to v1.45.0, kin-openapi to v0.149.0, cel-go to v0.29.0, golang.org/x/crypto to v0.55.0 and the rest of the golang.org/x/* set, by moving pushgateway to 1.11.3 and image-availability-exporter to 0.16.0, and by building the log-shipper reloader with Go 1.25.13; advisories without a reachable fix are mitigated via VEX.
- **[deckhouse-controller]** Stop the module source registry fan-out from replaying and flooding the main queue with duplicate moduleRun tasks.
- **[deckhouse-controller]** Do not commit the package repository registry checksum when no application could be annotated, which left them all on stale registry settings.
- **[descheduler]** Fix duplicate enabled key in doc-ru-deschedulers.yaml
- **[docs]** Fixed FAQ section for ansible inventory.
- **[istio]** Fix istiod RBAC rules that drifted away from upstream.
- **[log-shipper]** Fixed an invalid VRL script generated by the `replaceValue` transformation. log-shipper
- **[node-manager]** Stop caps-controller-manager from rewriting StaticInstance objects in a hot loop when a connectivity check fails, which caused a sustained load on etcd.
  A StaticInstance whose connectivity check fails now keeps its reservation for the whole bootstrap window instead of returning to the pool on every failed attempt, and the bootstrap (20 min) and cleanup (10 min) timeouts, which previously never fired, are now reachable. If nothing ever ran on the host, the bootstrap timeout returns the instance to the pool after 20 minutes as before. If the host was already bootstrapped in part, the instance returns only through MachineHealthCheck remediation (nodeStartupTimeoutSeconds: 1200) plus the cleanup timeout, so it becomes available again roughly 20 + 10 minutes after the failure, and remediation reboots the host as part of the cleanup.
- **[tools]** Bump d8-cli to v0.32.6 - `d8 mirror push` restores the release channel tags of the installer image, so `docker run <repo>/install:<channel>` works against a mirrored registry
- **[tools]** Bump d8-cli to v0.32.7 - closes the CVE findings in the bundled d8 binary (grpc 1.83.2, x/image 0.45.0, x/crypto 0.55.0 and x/mod 0.40.0)
- **[upmeter]** Fixed ListenerSet upmeter namespace.
- **[user-authn]** Render a valid Deployment for a DexAuthenticator whose `resources` limits are fractional or below the init container requests.
  The `self-signed-generator` init container now carries the fixed `10m` / `25Mi` it was always meant to have, the same as the identical init container of `user-api`, instead of the sum of the main container limits. A DexAuthenticator that set large limits no longer gets that sum as burst: generating the self-signed certificate then takes 3 to 4 seconds instead of under a second. It happens once per pod start and needs no action.

1.76.14

Know before update

- D8UserAuthzRenderSlow requires at least three renders averaging above 30 seconds over 30 minutes, D8UserAuthzReleaseSlow at least two releases averaging above 5 minutes over one hour, and both must hold for 15 minutes. Severities are one step lower (6 and 7). Their descriptions now point at CPU shortage on the node rather than at the number of ClusterAuthorizationRules, which does not affect render time.
- During the first `user-authz` release after the update the per-role bindings are protected from the release engine, the aggregated ones are created, and the per-role bindings are deleted right after the release. Permissions granted through annotated ClusterRoles stay in place throughout. The `user-authz.deckhouse.io/access-level` label is now set automatically on annotated ClusterRoles. In audit logs, access granted through annotated ClusterRoles is attributed to `user-authz:<level>:custom` instead of the individual ClusterRole.
- The `service-with-healthchecks` controller and agent are restarted. A `ServiceWithHealthchecks` that was left in `NotAllEndpointsAreReady` with no EndpointSlice recovers automatically once the updated agent starts, no manual action is needed.

Features

- **[docs]** Add extended-monitoring module architecture section.
- **[docs]** Add registry-packages-proxy module architecture section.

Fixes

- **[service-with-healthchecks]** Fixed a load balancer losing all its endpoints for good after a

target pod was replaced.
The `service-with-healthchecks` controller and agent are restarted. A `ServiceWithHealthchecks` that was left in `NotAllEndpointsAreReady` with no EndpointSlice recovers automatically once the updated agent starts, no manual action is needed.
- **[user-authn]** Fixed DexAuthenticator pod creation under ResourceQuota by setting init container CPU/memory limits to the sum of main container limits.
- **[user-authz]** Grant custom ClusterRoles (annotated with `user-authz.deckhouse.io/access-level`) through one aggregated ClusterRole per access level, so the number of bindings per ClusterAuthorizationRule/AuthorizationRule no longer depends on the number of such roles. During the first `user-authz` release after the update the per-role bindings are protected from the release engine, the aggregated ones are created, and the per-role bindings are deleted right after the release. Permissions granted through annotated ClusterRoles stay in place throughout. The `user-authz.deckhouse.io/access-level` label is now set automatically on annotated ClusterRoles. In audit logs, access granted through annotated ClusterRoles is attributed to `user-authz:<level>:custom` instead of the individual ClusterRole.
- **[user-authz]** Group the module alerts under a group named apart from the alerts themselves, so the incident shelf accepts them.
- **[user-authz]** The D8UserAuthzRenderSlow and D8UserAuthzReleaseSlow alerts no longer fire on a single slow render or release, such as a cluster bootstrap under CPU pressure or the one-time migration pass.
  D8UserAuthzRenderSlow requires at least three renders averaging above 30 seconds over 30 minutes, D8UserAuthzReleaseSlow at least two releases averaging above 5 minutes over one hour, and both must hold for 15 minutes. Severities are one step lower (6 and 7). Their descriptions now point at CPU shortage on the node rather than at the number of ClusterAuthorizationRules, which does not affect render time.

1.76.13

Know before update

- A Deckhouse release suspended on its release channel is no longer applied by clusters that are behind and reach it through a step-by-step update. The suspend flag lives only in the release-channel image; previously it was dropped when the target release was built from its per-version image, so lagging clusters updated to a suspended release anyway.
- After the update, the controller adds the annotations and labels of every ServiceWithHealthchecks to the Service created for it. If a ServiceWithHealthchecks of the `LoadBalancer` type carries annotations that configure the load balancer (`network.deckhouse.io/load-balancer-ips`, `network.deckhouse.io/load-balancer-shared-ip-key` and similar), the load balancer controller applies them and may assign a different address to the service, which causes a short interruption of the connections.
- Endpoints for pods in a terminal phase (Failed/Succeeded) are no longer published. In DVP clusters this prevents traffic from being routed to a VirtualMachine IP that has been reused by another pod. Pods being deleted are now published with the serving and terminating conditions, which enables the graceful shutdown flow for consumers. Pod readiness is derived from the PodReady condition, and stale probe results are reset when a pod becomes not ready, is recreated, or changes its IP.

- The SSH key and sudo password in `SSHCredentials` (used by CAPS to reach static nodes) are now returned as `<omitted>` to callers without `get` on the `sshcredentials/sensitive` subresource, masked in the audit log, and encrypted in etcd when `apiserver.encryptionEnabled` is on. `d8:manage:infrastructure:viewer` and `:manager` no longer see these values. CAPS still does, and so does every holder of a `deckhouse.io` wildcard, since `*` matches subresources: `SuperAdmin`, the `kubeadm:cluster-admins` group, and the `deckhouse` and `webhook-handler` SAs of `d8-system`.

  For masked readers the `last-applied-configuration` annotation disappears from `get -o yaml`, and creating an `SSHCredentials` with `<omitted>` fails with `422 Invalid` — editing still works. Treat previously exposed keys as leaked and rotate them. With `apiserver.encryptionEnabled` on, rewrite existing objects to encrypt them: `d8 k get sshcredentials -o json | d8 k replace -f -`.

- The `descheduler` Deployment in `d8-descheduler` restarts once on upgrade to pick up the new argument. Descheduling behavior is unchanged for workloads that do not implement their own eviction policy.
- The `service-with-healthchecks` status logic was heavily refactored to reduce API and etcd load. If you rely on `lastProbeTime` observability on every probe, explicitly enable `verboseStatus` in the module configuration.
- The installer, the deckhouse ServiceAccount, holders of the cluster administrator kubeconfig and the kube-system / d8-system ServiceAccounts can create a User or a Group named by an existing authorization rule without the acknowledgement annotation. Every other requester is checked as before.

Features

- **[descheduler]** Always run the descheduler with the upstream `EvictionsInBackground` feature gate enabled.
  The `descheduler` Deployment in `d8-descheduler` restarts once on upgrade to pick up the new argument. Descheduling behavior is unchanged for workloads that do not implement their own eviction policy.
- **[docs]** Add managed-valkey module architecture section.
- **[docs]** Add monitoring-kubernetes module architecture section.
- **[docs]** Add documentation for Managed PostgreSQL service.

Fixes

- **[ci]** Add REGCTL_CONFIG
- **[ci]** Fix regctl in 1.76 build.
- **[common]** Quote user-controlled values in the prometheus, upmeter, prometheus-pushgateway and extended-monitoring templates to prevent YAML injection into the module release.
- **[deckhouse]** validate the hardlink target when extracting module and package images
- **[deckhouse-controller]** Honor a channel-level release suspend for clusters that reach the suspended version through a step-by-step update.
  A Deckhouse release suspended on its release channel is no longer applied by clusters that are behind and reach it through a step-by-step update. The suspend flag lives only in the release-channel image; previously it was dropped when the target release was built from its per-version image, so lagging clusters updated to a suspended release anyway.
- **[deckhouse-controller]** Fix tests for check_release_tests
- **[docs]** Documented the identity collision rules for local Group and User objects and the allow-access-to-kubernetes annotation, and corrected the described scope of a security subsystem manage role.
- **[docs]** add static cluster OS change for master nodes and Stronghold backup alerts
- **[node-manager]** Fix NodeCapacity calculation.

- **[node-manager]** Hide the CAPS SSH key and sudo password in SSHCredentials from users without the `sshcredentials/sensitive` subresource.
  The SSH key and sudo password in `SSHCredentials` (used by CAPS to reach static nodes) are now returned as `<omitted>` to callers without `get` on the `sshcredentials/sensitive` subresource, masked in the audit log, and encrypted in etcd when `apiserver.encryptionEnabled` is on. `d8:manage:infrastructure:viewer` and `:manager` no longer see these values. CAPS still does, and so does every holder of a `deckhouse.io` wildcard, since `*` matches subresources: `SuperAdmin`, the `kubeadm:cluster-admins` group, and the `deckhouse` and `webhook-handler` SAs of `d8-system`.

  For masked readers the `last-applied-configuration` annotation disappears from `get -o yaml`, and creating an `SSHCredentials` with `<omitted>` fails with `422 Invalid` — editing still works. Treat previously exposed keys as leaked and rotate them. With `apiserver.encryptionEnabled` on, rewrite existing objects to encrypt them: `d8 k get sshcredentials -o json | d8 k replace -f -`.

- **[service-with-healthchecks]** Fixed an API server overload issue (“status storm”), resolved validation errors for ClusterIP services, corrected pod readiness evaluation logic, and improved code quality.
  The `service-with-healthchecks` status logic was heavily refactored to reduce API and etcd load. If you rely on `lastProbeTime` observability on every probe, explicitly enable `verboseStatus` in the module configuration.
- **[service-with-healthchecks]** Annotations and labels of a ServiceWithHealthchecks are now copied to the Service created for it.
  After the update, the controller adds the annotations and labels of every ServiceWithHealthchecks to the Service created for it. If a ServiceWithHealthchecks of the `LoadBalancer` type carries annotations that configure the load balancer (`network.deckhouse.io/load-balancer-ips`, `network.deckhouse.io/load-balancer-shared-ip-key` and similar), the load balancer controller applies them and may assign a different address to the service, which causes a short interruption of the connections.
- **[service-with-healthchecks]** Stopped publishing terminated pods in EndpointSlices and started publishing pods being deleted as terminating endpoints.
  Endpoints for pods in a terminal phase (Failed/Succeeded) are no longer published. In DVP clusters this prevents traffic from being routed to a VirtualMachine IP that has been reused by another pod. Pods being deleted are now published with the serving and terminating conditions, which enables the graceful shutdown flow for consumers. Pod readiness is derived from the PodReady condition, and stale probe results are reset when a pod becomes not ready, is recreated, or changes its IP.
- **[user-authz]** The identity collision check no longer applies to the platform’s own identities, so a retried bootstrap creates the initial User again instead of timing out.
  The installer, the deckhouse ServiceAccount, holders of the cluster administrator kubeconfig and the kube-system / d8-system ServiceAccounts can create a User or a Group named by an existing authorization rule without the acknowledgement annotation. Every other requester is checked as before.

1.76.12

Know before update

- The `vpa-recommender` pod is restarted and recommendations are recalculated with the new granularity. Memory recommendations will generally become lower (less over-provisioning), CPU recommendations become multiples of 10m. VPA objects with `updateMode: Auto/Recreate` may evict and recreate pods to apply the updated requests.
- ValidatingAdmissionPolicy reserved-public-hosts-* is removed. Ingresses that were denied because they matched publicDomainTemplate are allowed again. settings.reservedPublicHosts in ModuleConfig deckhouse remains accepted and is ignored. Enforcement returns in a later release.

Fixes

- **[deckhouse]** Stop reserving platform public hostnames at admission; keep the ModuleConfig field so existing settings still validate.
  ValidatingAdmissionPolicy reserved-public-hosts-* is removed. Ingresses that were denied because they matched publicDomainTemplate are allowed again. settings.reservedPublicHosts in ModuleConfig deckhouse remains accepted and is ignored. Enforcement returns in a later release.
- **[deckhouse]** Fixed goroutine and memory leak in upmeter-agent caused by per-request HTTP clients leaving idle keep-alive connections to Prometheus open forever.
- **[deckhouse-controller]** Fixed a bug when PackageRepository always used a HTTPS scheme.
- **[docs]** Add InPlaceOrRecreate to VPA modes list and move Auto to a warning alert.
- **[local-path-provisioner]** Add wildcard tolerations to the helper pod template so PVC provisioning works on tainted nodes after local-path-provisioner v0.0.32+.
- **[user-authn]** Commander releases older than 1.18 can grant Kubernetes API access on DexClient again.
- **[vertical-pod-autoscaler]** Reduce VPA memory recommendation rounding from 64Mi to 16Mi and round CPU recommendations up to 10m.
  The `vpa-recommender` pod is restarted and recommendations are recalculated with the new granularity. Memory recommendations will generally become lower (less over-provisioning), CPU recommendations become multiples of 10m. VPA objects with `updateMode: Auto/Recreate` may evict and recreate pods to apply the updated requests.

1.76.11

Know before update

- Containers attached through the `pods/ephemeralcontainers` subresource were not seen by the module's webhook, so no policy was evaluated for them: SecurityPolicy, OperationPolicy, Pod Security Standards and image signature verification were all bypassed by `kubectl debug`. After the update these requests are validated.

  On clusters with restricted Pod Security Standards, `deny-exec` or strict SecurityPolicy, typical debug images (root user, writable root filesystem, interactive `-it`) will now be rejected. Adjust the policies or use a SecurityPolicyException for the namespaces where debugging must stay available.

  The webhook uses `failurePolicy: Fail`, so while Gatekeeper is unavailable `kubectl debug` will not work in user namespaces. System namespaces are excluded from the webhook and are not affected.

- Creating a `Group` or a `User`, or renaming an existing one, so that `Group.spec.name` matches a group subject or `User.spec.email` matches a user subject of an existing AuthorizationRule or ClusterAuthorizationRule is now rejected, preventing an unintended privilege grant. The email is compared after lowercasing, since that is the form that reaches the token. Set the `user-authz.deckhouse.io/allow-authorization-rule-collision` annotation to `"true"` on the object if the collision is intentional. Already existing objects keep working and only produce a warning. Any declarative flow that manages both the rule and the identity is affected, not only one that deletes and recreates the identity: a first-time apply denies the `Group` or `User` whenever the rule happens to be applied first, and nothing guarantees the order. Add the annotation to such objects before upgrading. Deleting a `Group` or a `User` whose name is still a subject of a rule now produces a warning, since the rule keeps granting that name.

- On upgrade, every DexAuthenticator that carries the `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` annotation gets a freshly generated client secret, because it previously received the client secret of the privileged `kubernetes` OAuth2 client instead. Its pods roll once and users of those applications may have to sign in again. Each rotation is written to the Deckhouse log as `Rotating DexAuthenticator client secret that reused the shared kubernetes client secret`, with the authenticator name and namespace.

  If that annotation was ever used outside the `d8-*` and `kube-*` namespaces, the shared secret must be considered disclosed to anyone who could read Secrets in those namespaces, and has to be rotated by hand. Deleting the Secret alone does not rotate it: while the value is non-empty in

memory the owning hook re-renders the same one. Rotate it as follows.

1. If a GitOps tool syncs the `d8-user-authn` namespace, exclude `Secret/kubernetes-dex-client-app-secret` from that sync first, otherwise it is restored to the old value.
2. Blank the stored value: `kubectl -n d8-user-authn patch secret kubernetes-dex-client-app-secret --type merge -p '{"data":{"secret":""}}'`
3. Drop the in-memory copy: `kubectl -n d8-system rollout restart deployment/deckhouse`
4. Check that `kubectl -n d8-user-authn get secret kubernetes-dex-client-app-secret -o jsonpath='{.data.secret}'` now differs from the previous value. If it does not, the module re-rendered the old value before the restart took effect; repeat steps 2 and 3.

The blast radius of that rotation is wider than the `kubernetes` client alone. The same value is the client secret of the `kubeconfig-generator`, `kubeconfig-publish-api` and every `kubeconfig-<slug>` OAuth2 client, and it is passed to basic-auth-proxy as `--ldap-client-secret`. In-cluster consumers are re-rendered and their pods roll automatically, but kubeconfig files already downloaded from the kubeconfig generator carry the old client secret and can no longer refresh their tokens, so their users must download a new kubeconfig. Already issued ID tokens keep working until they expire, at most `settings.idTokenTTL`.

Deleting and recreating a DexAuthenticator is always a safe way to force a fresh per-authenticator secret: the credentials Secret is regenerated with the resource. Its content is owned by Deckhouse, so no GitOps change is needed for the per-authenticator rotation itself.

- Only a subject allowed to update the user-authn ModuleConfig, or the Deckhouse service account, may add the `dexclient.deckhouse.io/allow-access-to-kubernetes` or `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` annotation. Objects that already carry it keep working and stay editable by their owners as long as they are updated in place. A GitOps controller that deletes and recreates such an object instead of patching it will have the recreation denied, and the object will come back without access to the Kubernetes API unless the controller's service account is allowed to update the user-authn ModuleConfig.
- Project administrators (user-authz Admin access level) can no longer create, update or delete SecurityPolicyException resources; the permission now requires ClusterAdmin. Clusters that granted accessLevel Admin to untrusted tenants over RBAC v1 were exposed to node-level privilege escalation. RBAC v2 use/manage roles were never affected.
- Setting the annotation no longer requires write access to the user-authn ModuleConfig. Module administrators are unaffected, as their roles already carry the new permission, and so are the platform's own components in `d8-system` and `kube-system`. Any other subject that sets the annotation, a cluster provisioner in particular, has to be granted `create` on the subresource, which for a single namespace is:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: allow-access-to-kubernetes
  namespace: <namespace>
rules:
- apiGroups: ["deckhouse.io"]
  resources: ["dexclients/allow-access-to-kubernetes", "dexauthenticators/allow-access-to-
kubernetes"]
  verbs: ["create"]
```

  bound to the subject with a RoleBinding in the same namespace, or with a ClusterRole and a ClusterRoleBinding to grant it cluster-wide.
- The `d8:manage:permission:subsystem:kubernetes:view_resources` role no longer grants `nodes/proxy`, `nodes/log`, `nodes/metrics` and `nodes/stats`. Users holding `d8:manage:kubernetes:viewer`, `d8:manage:infrastructure:viewer` or any role aggregating them lose access to the kubelet API through `/api/v1/nodes/<node>/proxy/...`. `nodes` themselves stay readable. Automation that relied on reaching the kubelet API with one of these roles must be granted a dedicated role explicitly.
- `spec.oidc.scopes` items are now validated against `^\S+$` in both served versions of DexProvider. A scope containing whitespace could never work, because OAuth2 passes scopes as a space-delimited list, but it was previously admitted and silently broke authentication. Stored objects are not invalidated, however a DexProvider holding such a value is rejected on its next write until the value is corrected, which also affects a GitOps flow that reapplies the object unchanged.

Features

- **[docs]** Add previous page and next page buttons to modules documentation

Fixes

- **[admission-policy-engine]** Admission policies are now evaluated for ephemeral containers added with `kubectl debug`.
  Containers attached through the `pods/ephemeralcontainers` subresource were not seen by the module's webhook, so no policy was evaluated for them: SecurityPolicy, OperationPolicy, Pod Security Standards and image signature verification were all bypassed by `kubectl debug`. After the update these requests are validated.

  On clusters with restricted Pod Security Standards, `deny-exec` or strict SecurityPolicy, typical debug images (root user, writable root filesystem, interactive `-it`) will now be rejected. Adjust the policies or use a SecurityPolicyException for the namespaces where debugging must stay available.

  The webhook uses `failurePolicy: Fail`, so while Gatekeeper is unavailable `kubectl debug` will not work in user namespaces. System namespaces are excluded from the webhook and are not affected.

- **[admission-policy-engine]** Restrict SecurityPolicyException write access to ClusterAdmin, closing a tenant-to-node privilege escalation.
  Project administrators (user-authz Admin access level) can no longer create, update or delete SecurityPolicyException resources; the permission now requires ClusterAdmin. Clusters that granted accessLevel Admin to untrusted tenants over RBAC v1 were exposed to node-level privilege escalation. RBAC v2 use/manage roles were never affected.
- **[cloud-provider-yandex]** A Node that cannot be resolved to a Yandex Instance no longer blocks target group synchronization for the whole cluster.
- **[cloud-provider-zvirt]** Prevent capz-controller-manager from crashlooping without diagnostics when zVirt tags cannot be created.
- **[deckhouse]** bump addon-operator to run the module startup phase unconditionally
- **[deckhouse]** Deleted module documentation when a module is disabled.
- **[deckhouse]** The reserved public hosts policies no longer make every aggregated apiserver in the cluster watch all ConfigMaps, which left permission-browser-apiserver permanently unready and stopped namespace deletion cluster-wide.
- **[docs]** Fix the upmeter web UI Ingress name in SLA documentation
- **[testing]** Copy Helm render capabilities per render instead of mutating the shared defaults, and allow tests to declare available API kinds.
- **[user-authn]** Quote DexProvider-derived values rendered into the basic-auth-proxy Deployment and the Dex configuration file to prevent YAML injection.
- **[user-authn]** Reject whitespace in DexProvider OIDC scopes, which the OAuth2 protocol cannot represent.
  `spec.oidc.scopes` items are now validated against `^\S+$` in both served versions of DexProvider. A scope containing whitespace could never work, because OAuth2 passes scopes as a space-delimited list, but it was previously admitted and silently broke authentication. Stored objects are not invalidated, however a DexProvider holding such a value is rejected on its next write until the value is corrected, which also affects a GitOps flow that reapplies the object unchanged.

- **[user-authn]** Issue a dedicated OAuth2 client secret per DexAuthenticator instead of reusing the shared cluster client secret.
  On upgrade, every DexAuthenticator that carries the `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` annotation gets a freshly generated client secret, because it previously received the client secret of the privileged `kubernetes` OAuth2 client instead. Its pods roll once and users of those applications may have to sign in again. Each rotation is written to the Deckhouse log as `Rotating DexAuthenticator client secret that reused the shared kubernetes client secret`, with the authenticator name and namespace.

  If that annotation was ever used outside the `d8-*` and `kube-*` namespaces, the shared secret must be considered disclosed to anyone who could read Secrets in those namespaces, and has to be rotated by hand. Deleting the Secret alone does not rotate it: while the value is non-empty in memory the owning hook re-renders the same one. Rotate it as follows.

  1. If a GitOps tool syncs the `d8-user-authn` namespace, exclude `Secret/kubernetes-dex-client-app-secret` from that sync first, otherwise it is restored to the old value.
  2. Blank the stored value: `kubectl -n d8-user-authn patch secret kubernetes-dex-client-app-secret --type merge -p '{"data":{"secret":""}}'`
  3. Drop the in-memory copy: `kubectl -n d8-system rollout restart deployment/deckhouse`
  4. Check that `kubectl -n d8-user-authn get secret kubernetes-dex-client-app-secret -o jsonpath='{.data.secret}'` now differs from the previous value. If it does not, the module re-rendered the old value before the restart took effect; repeat steps 2 and 3.

  The blast radius of that rotation is wider than the `kubernetes` client alone. The same value is the client secret of the `kubeconfig-generator`, `kubeconfig-publish-api` and every `kubeconfig-<slug>` OAuth2 client, and it is passed to basic-auth-proxy as `--ldap-client-secret`. In-cluster consumers are re-rendered and their pods roll automatically, but kubeconfig files already downloaded from the kubeconfig generator carry the old client secret and can no longer refresh

their tokens, so their users must download a new kubeconfig. Already issued ID tokens keep working until they expire, at most `settings.idTokenTTL`.

Deleting and recreating a DexAuthenticator is always a safe way to force a fresh per-authenticator secret: the credentials Secret is regenerated with the resource. Its content is owned by Deckhouse, so no GitOps change is needed for the per-authenticator rotation itself.

- **[user-authn]** Drop the dead per-DexAuthenticator redirect URI from the privileged kubernetes OAuth2 client.
- **[user-authn]** Adding the allow-access-to-kubernetes annotation to a DexClient or a DexAuthenticator now requires cluster-level authority over the user-authn module.
  Only a subject allowed to update the user-authn ModuleConfig, or the Deckhouse service account, may add the `dexclient.deckhouse.io/allow-access-to-kubernetes` or `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` annotation. Objects that already carry it keep working and stay editable by their owners as long as they are updated in place. A GitOps controller that deletes and recreates such an object instead of patching it will have the recreation denied, and the object will come back without access to the Kubernetes API unless the controller's service account is allowed to update the user-authn ModuleConfig.
- **[user-authn]** The policy gating the allow-access-to-kubernetes annotation no longer applies to platform components, and proves authority with a create permission on the dexclients/allow-access-to-kubernetes and dexauthenticators/allow-access-to-kubernetes subresources instead of write access to the user-authn ModuleConfig.
  Setting the annotation no longer requires write access to the user-authn ModuleConfig. Module administrators are unaffected, as their roles already carry the new permission, and so are the platform's own components in `d8-system` and `kube-system`. Any other subject that sets the annotation, a cluster provisioner in particular, has to be granted `create` on the subresource, which for a single namespace is:

  ```
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    name: allow-access-to-kubernetes
    namespace: <namespace>
  rules:
  - apiGroups: ["deckhouse.io"]
    resources: ["dexclients/allow-access-to-kubernetes", "dexauthenticators/allow-access-to-
  kubernetes"]
    verbs: ["create"]
  ```

  bound to the subject with a RoleBinding in the same namespace, or with a ClusterRole and a ClusterRoleBinding to grant it cluster-wide.
- **[user-authz]** Reject local Group names and User emails that collide with a subject already granted privileges by an authorization rule.
  Creating a `Group` or a `User`, or renaming an existing one, so that `Group.spec.name` matches a group subject or `User.spec.email` matches a user subject of an existing AuthorizationRule or ClusterAuthorizationRule is now rejected, preventing an unintended privilege grant. The email is compared after lowercasing, since that is the form that reaches the token. Set the `user-authz.deckhouse.io/allow-authorization-rule-collision` annotation to `"true"` on the object if the collision is intentional. Already existing objects keep working and only produce a warning. Any declarative flow that manages both the rule and the identity is affected, not only one that deletes and recreates the identity: a first-time apply denies the `Group` or `User` whenever the rule happens to be applied first, and nothing guarantees the order. Add the annotation to such objects before upgrading. Deleting a `Group` or a `User` whose name is still a subject of a rule now produces a warning, since the rule keeps granting that name.
- **[user-authz]** Remove node subresources from the RBACv2 viewer permission role
  The `d8:manage:permission:subsystem:kubernetes:view resources` role no longer grants `nodes/proxy`, `nodes/log`, `nodes/metrics` and `nodes/stats`. Users holding `d8:manage:kubernetes:viewer`, `d8:manage:infrastructure:viewer` or any role aggregating them lose access to the kubelet API through `/api/v1/nodes/<node>/proxy/...`. `nodes` themselves stay readable. Automation that relied on reaching the kubelet API with one of these roles must be granted a dedicated role explicitly.

1.76.10

Know before update

- Fixes recreation of all CloudEphemeral nodes on upgrade to 1.76.9. The MachineDeployment replica count was dropped and the MachineDeployment was scaled to zero.
- Values taken from `Project.spec.parameters` are now quoted where they are substituted into the shipped project templates, and a parameter that changes the structure of the rendered manifests rather than only their values is refused for any template, including custom ones. An

administrator name may no longer contain control characters, and `clusterLogDestinationName` must be a Kubernetes object name or empty. Both are checked from the reconcile loop as well as at admission, so a project already carrying such a value goes into an error state on its next reconcile rather than at its next edit; the same holds for a custom template that turns a parameter into several objects or into YAML, which is refused when the project is created, edited, or reconciled with a change to apply. The module's admission policy now also matches CREATE, so objects labelled `heritage: multitenancy-manager` can only be created by the module itself.

- `d8:manage:permission:module:multitenancy-manager:edit` no longer grants write access to `projecttemplates`. A `ProjectTemplate` renders into a release applied by a ServiceAccount bound to `cluster-admin`, and its `spec.resourcesTemplate` is arbitrary, so authoring one is equivalent to holding `cluster-admin` and is not a module-level permission.

  Reading templates stays in `d8:manage:permission:module:multitenancy-manager:view`, and creating a `Project` on an existing template stays in the edit role. The legacy `ClusterAdmin` access level and the `kubeadm:cluster-admins` group keep the right. A subject that authored templates through the manager role has to be granted one of those instead.

- network-gateway pod's will be restarted

Features

- **[docs]** Add gpu module architecture section.
- **[docs]** Added cilium-hubble module architecture section.
- **[docs]** Added adaptive-resource-management module architecture section.
- **[docs]** Added operator-trivy module architecture section.
- **[docs]** Added managed-postgres and managed-memcached modules architecture sections.
- **[docs]** Add pod-reloader module architecture section.
- **[docs]** Add secrets-store-integration module architecture section.

Fixes

- **[cloud-provider-vcd]** Fixes an intermittent "network still in use" error during cluster deletion with the WithNAT placement strategy.
- **[deckhouse]** Fix values validation failing on the injected registry
- **[deckhouse-controller]** The module-documentation build request timeout to docs-builder is configurable via DOCUMENTATION_BUILD_TIMEOUT (default 120s) instead of a fixed 10s that canceled healthy full-site builds.
- **[docs]** Fix issues in managed-postgres and managed-memcached modules architecture sections.
- **[docs]** Clarify which integrity control mechanisms are available in which DKP editions and correct the DKP module integrity description.
- **[docs]** Link GPU driver install examples to gpu module FAQ
- **[docs]** Updated the Deckhouse Kubernetes Platform GUI installer links on the installation page from the `#gui-install` anchor to the `/gs/installer/` path.
- **[docs]** Corrected the documentation that incorrectly stated DKP supports only Bearer token authentication for container registries.
- **[docs]** Module documentation no longer gets stuck in Error under load; docs-builder builds are cancelable and no longer pile up redundant full-site rebuilds behind requests whose clients already timed out.
- **[istio]** Restore Istiod pod anti-affinity for Istio 1.25 installations managed by the Sail Operator.
- **[multitenancy-manager]** A project parameter can no longer inject objects into the rendered project manifests.
  Values taken from `Project.spec.parameters` are now quoted where they are substituted into the shipped project templates, and a parameter that changes the structure of the rendered manifests rather than only their values is refused for any template, including custom ones. An administrator name may no longer contain control characters, and `clusterLogDestinationName` must be a Kubernetes object name or empty. Both are checked from the reconcile loop as well as at admission, so a project already carrying such a value goes into an error state on its next reconcile rather than at its next edit; the same holds for a custom template that turns a parameter into several objects or into YAML, which is refused when the project is created, edited, or reconciled with a change to apply. The module's admission policy now also matches CREATE, so objects labelled `heritage: multitenancy-manager` can only be created by the module itself.

- **[multitenancy-manager]** The module manager role no longer grants authoring of project templates.
  `d8:manage:permission:module:multitenancy-manager:edit` no longer grants write access to `projecttemplates`. A `ProjectTemplate` renders into a release applied by a ServiceAccount bound to `cluster-admin`, and its `spec.resourcesTemplate` is arbitrary, so authoring one is equivalent to holding `cluster-admin` and is not a module-level permission.

  Reading templates stays in `d8:manage:permission:module:multitenancy-manager:view`, and creating a `Project` on an existing template stays in the edit role. The legacy `ClusterAdmin` access level and the `kubeadm:cluster-admins` group keep the right. A subject that authored templates through the manager role has to be granted one of those instead.

- **[network-gateway]** Fixed user and group in ConfigMap
  network-gateway pod's will be restarted
- **[node-manager]** Pin MachineDeployment minReadySeconds to 1 so the MachineSet apply converges.
- **[node-manager]** MachineDeployment `spec.replicas` is no longer dropped on upgrade, so cloud nodes are not recreated.
  Fixes recreation of all CloudEphemeral nodes on upgrade to 1.76.9. The MachineDeployment replica count was dropped and the MachineDeployment was scaled to zero.

1.76.9

Know before update

- The node-local-dns DaemonSet pods will restart after the update.

Features

- **[docs]** Add upmeter module architecture section.
- **[docs]** Add descheduler module architecture section.
- **[docs]** Add cni-cilium module architecture section.

Fixes

- **[cloud-provider-aws]** Do not fail the node bootstrap or update when `linux-modules-extra` is unavailable for the running kernel.
- **[deckhouse]** Locate CRDs by repository contents.
- **[deckhouse]** stop false drift detection reinstalling packages with helm hooks
- **[deckhouse]** An unset `settings.update.mode` now defaults to `AutoPatch` instead of silently running as `Auto`.
- **[deckhouse]** Fix a minor Deckhouse release being auto-applied as a patch when no Deployed release object is present.
- **[deckhouse]** limit the impact of a failed documentation build
- **[deckhouse]** Surface layer read errors instead of extracting a truncated image
- **[deckhouse]** Do not inject source registry into a still-embedded module
- **[deckhouse-controller]** Module releases rendered with nelm no longer raise false absent-resource alerts.
- **[deckhouse-controller]** nelm now takes over fields left by the old Helm 3 engine, so stale fields get removed on module upgrade.
- **[docs]** Clarify that DKP does not support version downgrades
- **[docs]** Add nil-safety guards for OpenAPI extraction to prevent panics.
- **[docs]** Fixed incorrect description of allowedHostPaths in SecurityPolicy documentation.
- **[docs]** Fix broken documentation links
- **[docs]** Clarified AutoPatch update mode and external module update settings.
- **[docs]** Fixed ModuleUpdatePolicy example in release channel FAQ.
- **[documentation]** Fix a memory leak in the documentation builder so memory is released between builds; add CPU/memory limits and an opt-in profiling endpoint.
- **[istio]** Add missing tools to proxyv2 images so the application-aware proxy termination hook works correctly.
- **[istio]** Fix graceful draining of established HTTP connections when application pods terminate.
- **[istio]** Create the ConfigMaps required for additional JWKS root CA certificates.
- **[keepalived]** Fixed Python linking in the keepalived module image.
- **[network-gateway]** Fixed Python linking in the network-gateway module image.
- **[node-local-dns]** Bump Go dependencies in the coredns helper image to fix known CVEs.
  The node-local-dns DaemonSet pods will restart after the update.
- **[registry]** Fix DKP bootstrap retry by keeping the `registry-init` secret.
- **[registrypackages]** Rebuild kubernetes-cni with updated Go dependencies to fix CVEs.
- **[tools]** Fixed module release validation incorrectly failing for `!optional` dependencies when the required module doesn't exist in the cluster.
- **[user-authn]** ClusterAdmin can watch the Dex storage CRDs (passwords, offlinesessionses), not only list them.

1.76.8

Know before update

- All Ingress-nginx controller pods will be restarted.
- The coredns and kube-dns components will restart after the update.
- The egress-gateway-agent component will restart after the update.
- The node-local-dns components will restart after the update.
- The service-with-healthchecks components (controller, agent) will restart after the update.

Features

- **[docs]** Update node-termination-handler behaviour in cloud-provider-aws module architecture section

Fixes

- **[cloud-provider-dvp]** skip foreign nodes in cloud controller manager
- **[cni-cilium]** Bump Go dependencies in the egress-gateway-agent image to fix known CVEs.
  The egress-gateway-agent component will restart after the update.
- **[deckhouse]** Skip the multiple CNI check in the enable_cni hook while a CNI migration is in progress to avoid a stuck Deckhouse queue.
- **[deckhouse-controller]** Re-derive the module release chain and align the from-to gap check with the release updater so intermediate releases are created instead of the chain freezing
- **[deckhouse-controller]** ModuleDocumentation will not be created for embedded modules.
- **[ingress-nginx]** Nginx version is updated to 1.30.4.
  All Ingress-nginx controller pods will be restarted.
- **[kube-dns]** Bump Go dependencies in the sts-pods-hosts-appender-webhook and coredns images to fix known CVEs.
  The coredns and kube-dns components will restart after the update.
- **[node-local-dns]** Bump Go dependencies in the safe-updater and stale-dns-connections-cleaner images to fix known CVEs.
  The node-local-dns components will restart after the update.
- **[service-with-healthchecks]** Bump Go dependencies in the service-with-healthchecks image to fix known CVEs.
  The service-with-healthchecks components (controller, agent) will restart after the update.
- **[user-authn]** Fix Dex token refresh with upstream providers that rotate refresh tokens (GitLab), which logged users out every `idTokenTTL`.
- **[user-authn]** backport fix of 21615
- **[user-authn]** Accessing kubernetes api via gateway api fixed.

1.76.7

Fixes

- **[cloud-provider-huaweicloud]** Fixed the Machine initialization contract — new nodes could not join the cluster.

1.76.6

Know before update

- All ingress-nginx controller pods of v1.10 will be restarted.
- The cilium-hubble components (hubble-ui, hubble-relay) will restart after the update.
- The cni-cilium components (cilium agent, operator) will restart after the update.
- The metallb components (controller, speaker, l2lb) will restart after the update.
- Users with accessLevel ClusterAdmin in ClusterAuthorizationRule gain read and write access to DVPInstanceClass objects; previously all access was denied.
- Users with accessLevel ClusterAdmin in ClusterAuthorizationRule gain read and write access to DynamixInstanceClass objects; previously all access was denied.

Fixes

- **[candi]** Add exception for Oracle Linux UEK kernels for fix CVE-2025-37999 on bashible
- **[cilium-hubble]** Upgrade hubble-ui backend dependencies (cilium v1.17.16, Go 1.25.0) and switch build base image to fix known CVEs.
  The cilium-hubble components (hubble-ui, hubble-relay) will restart after the update.
- **[cloud-provider-dvp]** Fix DVPInstanceClass access for users with user-authz ClusterAdmin role
  Users with accessLevel ClusterAdmin in ClusterAuthorizationRule gain read and write access to DVPInstanceClass objects; previously all access was denied.
- **[cloud-provider-dynamix]** Fix DynamixInstanceClass access for users with user-authz ClusterAdmin role
  Users with accessLevel ClusterAdmin in ClusterAuthorizationRule gain read and write access to DynamixInstanceClass objects; previously all access was denied.
- **[cloud-provider-vsphere]** fix missing default StorageClass annotation when a DatastoreCluster entry sorts before all Datastore entries
- **[cni-cilium]** Bump Go dependencies and backport upstream cilium security patches to fix known CVEs.
  The cni-cilium components (cilium agent, operator) will restart after the update.
- **[common]** Fix curl in debug-container by shipping a static binary
- **[common]** Bump Go dependencies in the cni-migration-controller and cni-migration-init-checker images to fix known CVEs.
- **[deckhouse]** atomically install modules and re-download incomplete versions
- **[deckhouse]** Restore admin access to list moduleconfigs
- **[deckhouse]** Fix hugo CVE.

- **[deckhouse-controller]** Fix maintenance state tracking — module no longer gets stuck in Unmanaged after removing maintenance.
- **[deckhouse-controller]** Force embedded modules back to the “Embedded” source on startup, healing a stale external source that previously stuck until the Module resource was deleted manually.
- **[docs]** Added a note about the termination of support for Ubuntu 20.04 and replace the link to the Docker container registry.
- **[docs]** Added a note about changing the name of the Docker container registry.
- **[documentation]** Bump Hugo version to 0.163.3
- **[ingress-nginx]** Controller 1.10 image uid and gid are set to deckhouse.
  All ingress-nginx controller pods of v1.10 will be restarted.
- **[metallb]** Bump Go dependencies in the metallb and l2lb images to fix known CVEs.
  The metallb components (controller, speaker, l2lb) will restart after the update.
- **[node-manager]** Fix create_rbac_and_certificate_for_kubernetes_api_proxy hook behavior.
- **[user-authn]** User-authn httproutes are refactored.

1.76.5

Know before update

- With enableMultiTenancy, effective access is now the union of CAR (within its limitNamespaces/namespaceSelector), AuthorizationRules, and plain RoleBindings/ClusterRoleBindings. Previously the webhook denied requests outside the CAR scope even when RBAC explicitly granted them: such existing bindings for subjects with a CAR silently become effective after the upgrade — review them. The CAR access level still does not apply outside its namespace limits. AccessibleNamespaces reflects the same union.
- log-shipper

Fixes

- **[cert-manager]** Additional letsencrypt-staging cluster issuer for Gateway API is provided.
- **[ci]** This is a manual cherry-pick the script for the CI fix in the release branch
- **[cilium-hubble]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[cloud-provider-dvp]** fix dvp kubernetes dependency mismatch
- **[cloud-provider-huaweicloud]** Adds patches to the upstream version to make it ignore static nodes
- **[deckhouse]** Bump addon-operator to fix module install with conversion webhooks
- **[deckhouse-controller]** Bump addon-operator to v1.21.22 — propagate module context to nelm logs.
- **[deckhouse-tools]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[docs]** Fix ingressClass description in global settings
- **[docs]** Corrected the link from direct to relative in the alert about the module requirements.
- **[docs]** Corrected the link from direct to relative in the alert about the module requirements for external modules.
- **[docs]** Add missing sidebar title mappings for Istio documentation CR page.
- **[documentation]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[istio]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[log-shipper]** fix daemonset template
  log-shipper
- **[openvpn]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[prometheus]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[registry]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[registry-packages-proxy]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[upmeter]** Gateway API certificate manifest is updated to support any cluster issuer.
- **[user-authn]** Gateway API certificate manifest is updated to support any cluster issuer.
  Kubernetes API HTTProute was updated to return 403 instead of 404.
- **[user-authz]** Honor CAR-independent RBAC in webhook and permission-browser
  With enableMultiTenancy, effective access is now the union of CAR (within its limitNamespaces/namespaceSelector), AuthorizationRules, and plain RoleBindings/ClusterRoleBindings. Previously the webhook denied requests outside the CAR scope even when RBAC explicitly granted them: such existing bindings for subjects with a CAR silently become effective after the upgrade — review them. The CAR access level still does not apply outside its namespace limits. AccessibleNamespaces reflects the same union.

1.76.4

Know before update

- kube-apiserver static pod is reconfigured and restarts on the next control-plane sync.

Features

- **[cloud-provider-dvp]** Add an option to disable the CCM LoadBalancer functionality via ConfigMap.
- **[docs]** Add operator-argo module architecture section
- **[docs]** Add module module architecture section

Fixes

- **[candi]** kube-apiserver no longer caches watches for `ManifestCheckpointContentChunk` resources from `state-snapshotter`.
  kube-apiserver static pod is reconfigured and restarts on the next control-plane sync.
- **[ci]** Fix extract images_digests.
- **[deckhouse]** Fix module metadata (`module.yaml`/`oss.yaml`) missing from in-cluster documentation by allowing these files in `IsDocsPath` (sender) and creating parent directories on extraction in the docs-builder (receiver).
- **[deckhouse]** Fixes an unauthenticated path traversal (CWE-22) in docs-builder by validating moduleName/channels and enforcing base-directory containment before any filesystem operation.
- **[deckhouse]** Remove documentation networkPolicy.
- **[deckhouse]** Add log diagnostic command to ModuleIsNotValid alert description.
- **[deckhouse]** Use metav1.Duration for ModulePullOverride scanInterval
- **[deckhouse-controller]** Fix deckhouse-controller crash loop and hooks receiving silently empty snapshots
- **[deckhouse-controller]** Fixed showing warnings while errors during kubectl edit.
- **[deckhouse-controller]** Fixed ModuleConfig validation.
- **[docs]** Document self-service password reset
- **[documentation]** remove stale fox links
- **[user-authn]** Preserve Dex Password fields (notably groups) when resetting password, locking or unlocking a user, so users are no longer locked out after these operations.

1.76.3

Know before update

- All ingress-nginx controllers pod will be restarted.
- Previously, a transient cluster DNS failure could cause the user-authz-webhook liveness probe to fail and restart the pod, which combined with the fail-closed authorization webhook (failurePolicy: Deny) could deny all API requests, including cluster-admins, until DNS recovered.

Features

- **[deckhouse-controller]** harden migrated modules check
- **[istio]** Custom annotations for Ingress Controller
- **[istio]** Custom extension providers
- **[istio]** JWKS extra root CA implementation

Fixes

- **[ci]** fix unlimited upmeter iterations
- **[cloud-provider-vcd]** add werf deploy-dependency annotations to capcd webhook configurations to fix DMT lint and prevent race on install/upgrade
- **[deckhouse]** Fix serialize docs builder builds.
- **[dhctl]** Preflight checks are no longer re-run on every dhctl-server restart when the cluster config is unchanged.
- **[docs]** Fix broken-link. Module admission
- **[docs]** Change security events docs
- **[ingress-nginx]** The config hashing is fixed.
  All ingress-nginx controllers pod will be restarted.
- **[kube-dns]** Added deploy dependencies for webhook configuration
- **[node-manager]** Creating or re-applying an already-existing StaticInstance no longer fails address validation.
- **[user-authz]** user-authz-webhook now uses the node-local kube-apiserver endpoint for its discovery cache and liveness check, instead of resolving the “kubernetes.default” DNS name. Previously, a transient cluster DNS failure could cause the user-authz-webhook liveness probe to fail and restart the pod, which combined with the fail-closed authorization webhook (failurePolicy: Deny) could deny all API requests, including cluster-admins, until DNS recovered.

1.76.2

Features

- **[docs]** Add registry module architecture section.
- **[istio]** Add deploy dependencies for Istio webhook configurations

Fixes

- **[deckhouse]** fix libs in the python-based images
- **[dhctl]** fix grpc stream cancel deadlock
- **[docs]** Added the ml/ai tag to the ai-models module on the modules homepage.
- **[registry]** Missing copy_custom_certificate for Gateway API path was added.
- **[registry-packages-proxy]** Missing Gateway API objects were added.
- **[vertical-pod-autoscaler]** Ensure VPA mutating webhook is deployed after its admission controller Deployment and Service are ready.

1.76.1

Know before update

- After upgrading to v1.76.0, kubectl logs and exec fail cluster-wide for all users. Manual workaround is available — see PR description.
- All ingress-nginx controller pods using default controller version will be restarted.

Features

- **[deckhouse]** Ensure conversion hooks.
- **[docs]** Add auxiliary components (of virtualization module) architecture page.
- **[docs]** Update csi-scsi-generic module architecture section
- **[docs]** Add deep-link anchors and auto-expand logic for collapsible sections.
- **[docs]** Add page about pod security settings

Fixes

- **[candi]** Fix containerd credential escaping by rendering username/password into auth string.
- **[ci]** remove CSE in deploy
- **[cloud-provider-dvp]** CVE fixes
- **[control-plane-manager]** fix Helm adoption of kubeadm:cluster-admins CRB on upgrade from pre-v1.76
  After upgrading to v1.76.0, kubectl logs and exec fail cluster-wide for all users. Manual workaround is available — see PR description.
- **[deckhouse]** Fix stale application conditions during update over a failed version
- **[deckhouse-controller]** Add alias for legacy bash completion flag.
- **[deckhouse-controller]** add werf dependency to webhook
- **[docs]** Deleted the note about the beta version of the graphical installer on the DKP installation page.
- **[docs]** Update the wording in the FAQ section
- **[ingress-nginx]** Default ingress controller version is reverted to 1.10.
  All ingress-nginx controller pods using default controller version will be restarted.
- **[user-authn]** Fix blank dex error page caused by missing BuildIdentifier in the error template data struct.

Version 1.75

Important

- Support for Kubernetes 1.35 has been added, while support for Kubernetes 1.30 has been discontinued. In future DKP releases, support for Kubernetes 1.31 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.33.

- Update to DKP 1.75 will be blocked if the cluster uses one of the following deprecated component versions (they are no longer supported):
  - IngressNginxController v1.9
  - Istio v1.19

- In DKP 1.75.0–1.75.3, the `PodSecurityStandardsViolation` alert may fire immediately after cluster installation. This happens when the `d8-system` namespace is missing the `heritage=deckhouse` label, causing Pod Security Standards policies to be mistakenly applied to DKP system pods. The issue is fixed in DKP 1.75.4.

  For clusters installed on DKP 1.75.0–1.75.3, add the required label manually using the following command:

  ```
  d8 k --as=system:sudouser label namespace d8-system heritage=deckhouse
  ```

- Istio version support changes:
  - Support for Istio 1.19 has been discontinued.
  - Istio 1.21 is now considered deprecated, and support for this version will be discontinued in the upcoming DKP releases. To update Istio, follow the instruction.

- Starting with Kubernetes 1.35, cluster nodes must use cgroup v2. The previous version (cgroup v1) is considered deprecated. If cgroup v2 is not supported on a node, the D8NodeCgroupV2NotSupported alert will be triggered. See the Kubernetes documentation for cgroup v2 migration details.
- Starting with Kubernetes 1.36, cluster nodes must use containerd v2 as the container runtime. Support for containerd v1 will be discontinued. If containerd v2 is not supported on a node, the D8NodeContainerdV2NotSupported alert will be triggered.
- The `vertical-pod-autoscaler` module has been updated from version 1.4.1 to 1.5.1. Starting from version 1.5.0, the `updateMode: Auto` mode is considered deprecated. Users are advised to replace it with `updateMode: InPlaceOrRecreate`. See the vertical scaling documentation for details.
- When issuing new certificates using the `cert-manager` module, the certificate's private key is now re-generated as well. This change mitigates potential security risks.
- Pods handling HTTP-01 challenges via Issuer and ClusterIssuer resources now have default resource requests defined. As a result, these pods consume part of the cluster and project resource quotas upon creation.
- The operator-trivy module is now delivered from an external source and follows an independent release cycle. No manual migration steps are required.

Deckhouse subsystem

- The ModuleSource resource now includes the `scanInterval` parameter to control how frequently the container registry is checked for new modules and versions.
- A dedicated node mode with the `node.deckhouse.io/etcd-arbiter` label has been introduced. In this mode, only etcd runs on the node (no control plane components). This enables a highly available cluster with two master nodes plus a dedicated etcd node, reducing resource requirements compared to the traditional three full master nodes configuration. See the High Availability documentation for details.
- New operating modes have been introduced in the `registry` module: **Proxy** (with image caching) and **Local** (for isolated environments). These modes are currently supported only for already deployed static clusters and are not yet available during DKP cluster installation.
- Container registry access parameters can now be configured via the deckhouse module during DKP cluster installation. Support for the DKP cluster deployment via Direct and Unmanaged modes has been added. The Legacy Unmanaged mode (via InitConfiguration) is preserved for compatibility. If no explicit configuration is provided, the mode is selected automatically.
- Added user shell validation: if a shell other than bash is used, the `dhctl` utility now displays a clear error message.
- Added StaticInstance validation before DKP installation using the corresponding SSHCredentials, preventing `dhctl` from reporting readiness before the cluster deployment is actually complete.
- Extended preflight test coverage to improve cluster deployment predictability.
- Partially reduced usage of netcat in bashible, lowering the number of security system triggers.
- Fixed an issue where cluster installation could not be resumed after interruption.
- Fixed Node object deletion logic during `converge`. The node is now properly removed from the cluster after the corresponding VM is destroyed.
- Fixed the `--skip-resources` flag behavior in `dhctl destroy` command. Specified resources are now correctly skipped.

Kubernetes & Scheduling subsystem

- A new `update-observer` component in the `control-plane-manager` module enables real-time tracking of control plane component updates (kube-apiserver, kube-scheduler, kube-controller-manager) and kubelet on nodes. This helps detect discrepancies and eliminates "blind spots" during Kubernetes updates.
- For all VPA resources managed by DKP, `updateMode: Auto` has been automatically replaced with `updateMode: InPlaceOrRecreate`. For clusters running Kubernetes 1.32 or earlier, the `InPlacePodVerticalScaling` feature gate is automatically enabled in control plane components (kube-apiserver, kube-scheduler, kube-controller-manager) and kubelet. This allows pod resource changes without recreation.

- The descheduler module has been updated to version 0.34. Key changes:
  - Priority-based pod eviction: pods with a higher restart count are evicted first, improving cluster stability during load redistribution.
  - DRA support: proper handling of resources managed via Dynamic Resource Allocation.
  - Default `cpu`, `memory`, and `pods` thresholds removed from `lowNodeUtilization` and `highNodeUtilization` strategies to align with upstream behavior.

IAM subsystem

- Namespace listing behavior has been changed. When requesting all cluster namespaces, users now see only the namespaces they have access to, even without permission to list all namespaces. This improves working with multitenancy and UI.
- The ProjectTemplate resource in the `multitenancy-manager` module now supports the unmanaged and skip-heritage options (set via annotations), allowing administrators to adjust the DKP control degree over resources:
  - `unmanaged`: The module does not manage the lifecycle of objects.
  - `skip-heritage`: DKP retains management but allows modification by external controllers.
- Added support for authenticating to the Kubernetes API via LDAP using Basic authentication instead of OIDC tokens (via the `ldap.enableBasicAuth` parameter in the DexProvider resource).
- The authorization webhook in the `user-authz` module now operates in fail-closed mode: if it's unavailable, authorization is denied. For improved reliability, webhook communication with the apiserver is performed via the cluster node network.
- The `user-authn` module now supports a global user session lifetime limit (via `refreshTokenAbsoluteLifetime`). Re-authentication is required after expiration.
- The `user-authn` module now supports Kerberos (SPNEGO) authentication in DKP.
- Updated kube-apiserver TLS cipher suites. Removed the following insecure ciphers from the `--tls-cipher-suites` flag: TLS_RSA_WITH_AES_256_GCM_SHA384 and TLS_RSA_WITH_AES_128_GCM_SHA256 (considered unsafe due to RSA key exchange). Added TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 and TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256.
- The anonymous access restriction mechanism for the API server has been modified. Instead of using the `--anonymous-auth=false` flag and a sidecar proxy, the AuthenticationConfiguration resource is now used to allow anonymous access only to `/livez`, `/readyz`, and `/healthz`. This improves cluster resource efficiency. Anonymous probe access remains restricted when public API access is enabled via `publishAPI.enabled`.

Security subsystem

- Fixed serialization of empty arrays and objects in SecurityPolicy and OperationPolicy resources of the `admission-policy-engine` module. Previously, some settings could be applied incorrectly.
- Revised ModSecurity extension implementation. OWASP Core Rule Set (CRS) functionality is now available in all supported IngressNginxController versions.

Cluster & Infrastructure subsystem

- Lifecycle stages updated for cloud provider integration modules:
  - `cloud-provider-dvp` (DVP), `cloud-provider-vcd` (VMware Cloud Director), `cloud-provider-huaweicloud` (Huawei Cloud), and `cloud-provider-zvirt` (zVirt) modules are now General Availability (previously Experimental).
- Added support for swap memory usage by pods. Instead of immediate OOM-kill, a pod can temporarily use swap (if configured and available on the node) and continue running after peak usage. Requires cgroup v2 and sufficient node resources. For details, see the `kubelet.memorySwap` parameter in the NodeGroup resource documentation.
- In the DVP cloud provider:
  - Added the `networkPolicy` parameter to define network traffic rules for workloads within a Project resource.
  - Added `managed-by`, `cluster-uuid`, and `vm_name` labels to all infrastructure objects.
  - Improved CSI driver diagnostics. Diagnostic messages are now displayed directly in PersistentVolumeClaim status.
  - Implemented automatic cleanup of VMBDA objects when a VM is deleted.
  - Fixed an issue where DeckhouseMachine objects could remain in deleting state after VM removal.
  - Fixed an issue where Cloud Controller Manager (CCM) could recreate external load balancers

after the associated Kubernetes Service was deleted.

- In Yandex Cloud, fixed a false NAT warning displayed during installation in configurations where NAT is not used.
- In the `node-manager` module:
  - Cluster API (CAPI) has been updated from 1.10.6 to 1.11.3, improving stability and provider compatibility.
  - Added a dedicated controller for NodeGroup hooks (extracted from DKP), reducing load on master nodes and improving responsiveness and UI feedback.
  - Added per-GPU MIG configuration support. You can now define a distinct set of MIG partitions for each physical GPU on a node. Previously, MIG configuration was applied at the node level and identical across all GPUs. This enables more flexible workload resource allocation on a single node.

Network subsystem

- In IngressNginxController, the `acceptClientIPHeadersFrom` parameter has been added for the following inlets: HostPort, HostPortWithProxyProtocol, LoadBalancer, and LoadBalancerWithProxyProtocol. The parameter defines trusted CIDR ranges for `X-Forwarded-*` headers that should not be overwritten during proxying.
- In the `cni-cilium` module, network observability capabilities have been extended:
  - Added extended metrics for monitoring interactions between specific objects (Pods, Services, Namespaces), configurable via the `extendedMetrics` parameter of the HubbleMonitoringConfig resource.
  - Implemented export of traffic flow logs for analyzing network interactions, configurable via the `flowLogs` parameter of the HubbleMonitoringConfig resource.
- Fixed an issue where, after VM migration in DVP, a LoadBalancer IP address could stop being announced when using MetalLB in BGP mode.
- IngressNginxController version support changes:
  - Support for v1.9 has been discontinued.
  - Support for v1.14 has been added.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.75.16

Fixes

- **[istio]** Add missing tools to proxyv2 images so the application-aware proxy termination hook works correctly.
- **[istio]** Fix graceful draining of established HTTP connections when application pods terminate.

1.75.15

Know before update

- The cilium-hubble components (hubble-ui, hubble-relay) will restart after the update.
- The cni-cilium components (cilium agent, operator) will restart after the update.
- The coredns and kube-dns components will restart after the update.
- The egress-gateway-agent component will restart after the update.
- The metallb components (controller, speaker, l2lb) will restart after the update.
- The node-local-dns DaemonSet pods will restart after the update.
- The node-local-dns components will restart after the update.
- The service-with-healthchecks components (controller, agent) will restart after the update.

Fixes

- **[cilium-hubble]** Upgrade hubble-ui backend dependencies (cilium v1.17.16, Go 1.25.0) and switch build base image to fix known CVEs.
  The cilium-hubble components (hubble-ui, hubble-relay) will restart after the update.
- **[cni-cilium]** Bump Go dependencies in the egress-gateway-agent image to fix known CVEs.
  The egress-gateway-agent component will restart after the update.
- **[cni-cilium]** Bump Go dependencies and backport upstream cilium security patches to fix known CVEs.
  The cni-cilium components (cilium agent, operator) will restart after the update.
- **[deckhouse]** An unset `settings.update.mode` now defaults to `AutoPatch` instead of silently running as `Auto`.
- **[deckhouse]** Fix a minor Deckhouse release being auto-applied as a patch when no Deployed release object is present.

- **[deckhouse]** Surface layer read errors instead of extracting a truncated image
- **[kube-dns]** Bump Go dependencies in the sts-pods-hosts-appender-webhook and coredns images to fix known CVEs.
  The coredns and kube-dns components will restart after the update.
- **[metallb]** Bump Go dependencies in the metallb and l2lb images to fix known CVEs.
  The metallb components (controller, speaker, l2lb) will restart after the update.
- **[node-local-dns]** Bump Go dependencies in the safe-updater and stale-dns-connections-cleaner images to fix known CVEs.
  The node-local-dns components will restart after the update.
- **[node-local-dns]** Bump Go dependencies in the coredns helper image to fix known CVEs.
  The node-local-dns DaemonSet pods will restart after the update.
- **[service-with-healthchecks]** Bump Go dependencies in the service-with-healthchecks image to fix known CVEs.
  The service-with-healthchecks components (controller, agent) will restart after the update.
- **[tools]** Fixed module release validation incorrectly failing for `!optional` dependencies when the required module doesn't exist in the cluster.

1.75.14

Fixes

- **[common]** Fix curl in debug-container by shipping a static binary
- **[deckhouse]** atomically install modules and re-download incomplete versions
- **[deckhouse]** show the allow-disabling annotation hint when disabling a confirmation-required module
- **[deckhouse]** Fix CVEs.
- **[deckhouse-controller]** Fix maintenance state tracking — module no longer gets stuck in Unmanaged after removing maintenance.
- **[docs]** Corrected the link from direct to relative in the alert about the module requirements.

1.75.13

Fixes

- **[deckhouse]** Fix nil pointer panic in resource and namespace informers by using each informer's own logger instead of the possibly-nil Monitor.Logger.
- **[user-authn]** Preserve Dex Password fields (notably groups) when resetting password, locking or unlocking a user, so users are no longer locked out after these operations.

1.75.12

Know before update

- Previously, a transient cluster DNS failure could cause the user-authz-webhook liveness probe to fail and restart the pod, which combined with the fail-closed authorization webhook (failurePolicy: Deny) could deny all API requests, including cluster-admins, until DNS recovered.

Fixes

- **[candi]** fix if node has bashible-uninitialized taint in race condition.
- **[deckhouse]** add cryptography CVE VEX to deckhouse-controller
- **[deckhouse]** Fix module metadata (`module.yaml` / `oss.yaml`) missing from in-cluster documentation by allowing these files in `IsDocsPath` (sender) and creating parent directories on extraction in the docs-builder (receiver).
- **[deckhouse-controller]** Don't create an external module release for a module that is still shipped embedded, so it can't replace or duplicate the embedded copy.
- **[deckhouse-controller]** Disabling modules with confirmation will be rejected without using the annotation.
- **[docs]** Added the ml/ai tag to the ai-models module on the modules homepage.
- **[docs]** Fix broken-link. Module admission
- **[documentation]** Add /app/hugo to docs-builder mount points
- **[user-authz]** user-authz-webhook now uses the node-local kube-apiserver endpoint for its discovery cache and liveness check, instead of resolving the "kubernetes.default" DNS name. Previously, a transient cluster DNS failure could cause the user-authz-webhook liveness probe to fail and restart the pod, which combined with the fail-closed authorization webhook (failurePolicy: Deny) could deny all API requests, including cluster-admins, until DNS recovered.

1.75.11

Know before update

- All ingress-nginx pods will be restarted.

Fixes

- **[common]** Fixed CVE-2026-40898 in CoreDNS by updating the quic-go dependency.
- **[ingress-nginx]** Nginx is updated to 1.30.3.
  All ingress-nginx pods will be restarted.

1.75.10

Features

- **[docs]** Add csi-ceph module architecture section.
- **[docs]** Add Core component (of virtualization module) architecture section.
- **[documentation]** Update editions availability info for stronghold module.

Fixes

- **[candi]** fix cve node-manager and opentofu.
- **[cilium-hubble]** Fixed CVE-2026-29181 in hubble-ui-backend by bumping OpenTelemetry Go to v1.41.0
- **[cilium-hubble]** Fixed CVE-2026-41520 in hubble-ui-backend
- **[cni-cilium]** Fixed CVE-2026-41520 for cilium-bugtool util
- **[cni-flannel]** Reverted module stage from Deprecated back to General Availability to stop false deprecation alerts.
- **[common]** fixed CVE-2026-29181 in the CoreDNS
- **[deckhouse-controller]** A module that conditionally depends on another is no longer disabled when an incompatible version of that dependency is enabled; the enable is rejected instead.
- **[deckhouse-controller]** fix module config validation for disable warnings
- **[docs]** Add a general description of container registers DKP
- **[docs]** Updated the etcd restore documentation.
- **[docs]** Fix CRD version tabs markup and nesting.
- **[istio]** Implement graceful metadata secret renewal for multiclusters.
- **[network-policy-engine]** Reverted module stage from Deprecated back to General Availability to stop false deprecation alerts.
- **[node-manager]** hook to restore apiVersion on CAPI resources.
- **[node-manager]** Fix static NodeGroup readiness calculation.
- **[node-manager]** add rbac policies for persistantvolumes to manage from capi-controller-manager.
- **[tools]** fix d8 build

1.75.9

Know before update

- All Ingress-nginx controller pods will be restarted.

Fixes

- **[admission-policy-engine]** Refactor main documentation page
- **[ingress-nginx]** Nginx was updated to 1.30.2.
  All Ingress-nginx controller pods will be restarted.

1.75.8

Know before update

- Registry pods will be restarted.

Features

- **[docs]** Update deckhouse web modules architecture section.

Fixes

- **[csi-vsphere]** Fixed the Deckhouse queue getting stuck
- **[docs]** Change network-policy-engine and cni-flannel to deprecated.
- **[monitoring-deckhouse]** Fix module-release alerts (`ModuleReleaseIsWaitingManualApproval`, `ModuleReleaseIsOutdated`, `ModuleReleaseIsBlockedByRequirements`, `ModuleIsInMaintenanceMode`, `D8ModuleOutdatedByMajorVersion`) that never fired on 1.75.x clusters due to a `moduleName` → `module` label mismatch introduced when addon-operator was bumped to v1.19.7 in v1.75.1.
- **[registry]** Updated auth image Go dependencies to fix Go CVEs.
  Registry pods will be restarted.
- **[user-authn]** Improve basic-auth-proxy request handling, cache implementation, and shutdown behavior.

1.75.7

Know before update

- All Ingress-nginx controller pods will be restarted.

Features

- **[docs]** Add Deckhouse web modules architecture section.
- **[docs]** Add cert-manager module architecture section.

Fixes

- **[cloud-provider-vsphere]** normalizes new paths and makes bashible resolve existing paths case-insensitively
- **[cni-cilium]** Fixed CVE-2026-35469
- **[deckhouse]** Fix exp modules auto enabling.
- **[deckhouse]** Fixed global configuration generation.
- **[dhctl]** Wait for stronghold cluster sync before node deletion
- **[docs]** Refactored web UI page.
- **[docs]** New screenshots have been added to the Console documentation.
- **[ingress-nginx]** Nginx is updated up to 1.30.1.
  All Ingress-nginx controller pods will be restarted.

1.75.6

Fixes

- **[admission-policy-engine]** Fixed enforcementAction for D8ReplicaLimits, added more template tests.
- **[dhctl]** fix SSH preflight check for StaticInstances with password-only auth.
- **[docs]** Fix broken links in VPA section.
- **[docs]** Fix visibility scope for nested details component
- **[docs]** Add info about kernel requirement for containerdv2 migration.
- **[node-manager]** CAPI crd served version fix
- **[node-manager]** Added cleanup for oversized MCM MachineSet revision history annotation

1.75.5

Know before update

- module pods will be restarted

Features

- **[docs]** Add a Jekyll plugin for tabbed content in the documentation.
- **[istio]** Allow custom ports in metadataEndpoint URLs for IstioFederation and IstioMulticluster CRDs.

Fixes

- **[candi]** Added a Netplan override to force the secondary NIC to use the main routing table, fixing cloud-init PBR conflicts.
- **[common]** Fixed an issue where Trivy could not detect VEX files in coredns images.
- **[deckhouse]** Overwrite currentReleaseImageName on mismatch.
- **[deckhouse]** Allow updating scanInterval on the deckhouse ModuleSource.
- **[istio]** fixed CVEs in module images
  module pods will be restarted
- **[service-with-healthchecks]** Correct RBACv2 is provided.
- **[terraform-manager]** Fix opentofu patches build after pull request 19080.

1.75.4

Know before update

- All Ingress-NGINX controller pods will be restated.
- All pods of Ingress-NGINX controller of 1.10 and 1.12 versions will be restarted.
- Cilium agents will be restarted during the update.
- HPA scaler pods will be restarted.
- Kruise controller will be restarted.
- These changes prevent unnecessary or destructive plan updates that could occur when data sources depend on changing labels and annotations. The behavior of other cloud providers is not affected. If you encounter unexpected converge plans or cluster bootstrap issues when using OpenTofu-based providers (such as dvp, dynamics, zvirt, or yandex), please report them to

Deckhouse Technical Support.
- This update triggers a rolling update of the flannel pods.
- This update triggers a rolling update of the kube-proxy pods.
- This update triggers a rolling update of the network-policy-engine pods.
- istio module pods will be restarted
- istio-cni-nodes will be restarted

Features

- **[docs]** Respect nowrap default for code blocks.
- **[docs]** Update sds-node-configuration module architecture section.
- **[docs]** Add geoproxy and failover to ingress-nginx module architecture section
- **[docs]** Add csi-nfs module architecture section.
- **[docs]** Add public cloud providers aws, azure, and gcp module architecture section.
- **[docs]** Add cloud-provider-vsphere module architecture section.
- **[docs]** Add cloud-provider-openstack module architecture section.
- **[terraform-manager]** Suppress destructive changes when add/remove/change labels and annotations for cloud resources via opentofu.
  These changes prevent unnecessary or destructive plan updates that could occur when data sources depend on changing labels and annotations. The behavior of other cloud providers is not affected. If you encounter unexpected converge plans or cluster bootstrap issues when using OpenTofu-based providers (such as dvp, dynamics, zvirt, or yandex), please report them to Deckhouse Technical Support.

Fixes

- **[admission-policy-engine]** Fix high resource consumption for constraint d8denyexecheritage
- **[candi]** Fix containerd unmount for migration if /var/lib/containerd is mounted manually.
- **[cert-manager]** Disable SecurityPolicyExceptions for cert-manager namespace
- **[cloud-provider-aws]** add information about AWS security group rules limits
- **[cloud-provider-dvp]** Suppress destructive changes when add/remove/change labels and annotations for cloud resources via opentofu.
- **[cloud-provider-vcd]** fix vCD CCM TCP health monitors removal
- **[cni-cilium]** Updated go-jose dependency to v4.1.4 to fix CVE-2026-34986.
  Cilium agents will be restarted during the update.
- **[cni-flannel]** Fixed CVE-2026-33186 by updating google.golang.org/grpc in flanneld.
  This update triggers a rolling update of the flannel pods.
- **[cni-simple-bridge]** Refactored python image source and pip exclusion.
- **[common]** Removed Python completely from the debug-container image as it is no longer needed, resolving corresponding CVEs, and silenced false positives for etcd binaries via VEX.
- **[deckhouse]** Return setting notified for mr.
- **[deckhouse]** Remove notified=false annotation reset from runReleaseDeploy in the module release controller.
- **[deckhouse]** Ensure heritage label on d8-system namespace via hook.
- **[deckhouse-controller]** Add new discoveryv1 scheme to deckhouse controller.
- **[deckhouse-controller]** added extra validation for kubernets version multiple downgrades scenario
- **[dhctl]** Use internal node ip if bastion ip was passed in converge.
- **[docs]** Fix indented Liquid blocks in lists.
- **[docs]** Fix broken links.
- **[docs]** Improved title generation for module documentation pages.
- **[ingress-nginx]** CVE-2025-15566 is fixed in 1.10 and 1.12 controllers.
  All pods of Ingress-NGINX controller of 1.10 and 1.12 versions will be restarted.
- **[ingress-nginx]** Added missing pod security context settings to hpa scaler pods.
  HPA scaler pods will be restarted.
- **[ingress-nginx]** Kruise go dependencies are updated to mitigate latest CVEs.
  Kruise controller will be restarted.
- **[ingress-nginx]** Added fix for CVE-2026-4342.
  All Ingress-NGINX controller pods will be restated.
- **[istio]** fixed CVE-2026-34986
  istio module pods will be restarted
- **[istio]** added iptables wrapper in cni-v1x21x6
  istio-cni-nodes will be restarted
- **[keepalived]** Excluded vulnerable pip-25.3 from keepalived final image to fix CVE-2026-1703
- **[kube-dns]** Refactored hook remove_old_coredns_deploy_and_svc — delegated deletion of original kube-dns service to helm.
- **[kube-proxy]** Fixed CVE-2026-33186 and CVE-2026-24051 in kube-proxy dependencies.
  This update triggers a rolling update of the kube-proxy pods.
- **[network-gateway]** Updated python image source and mitigated pip CVE-2026-1703
- **[network-policy-engine]** Fixed CVE-2026-34040, CVE-2026-33997, and CVE-2026-33186 in network-policy-engine dependencies.
  This update triggers a rolling update of the network-policy-engine pods.
- **[node-manager]** rollback inject capi conversion hook in crds as on-before-helm

- **[node-manager]** optimize node-controller cache resources usage.
- **[registrypackages]** Add vex for CVE-2026-39882
- **[registrypackages]** Remediated CVE-2026-32285 in crictl dependencies.

1.75.3

Know before update

- CoreDNS pods will undergo a rolling restart.

Features

- **[docs]** Add cloud-provider-yandex module architecture section.

Fixes

- **[common]** Fixed CVE-2026-33186 in the CoreDNS image.
  CoreDNS pods will undergo a rolling restart.
- **[deckhouse-controller]** revert nelm to helm
- **[dhctl]** Skip tmp lock for exporter and auto-converger.
- **[node-local-dns]** Fix werf manifest
- **[node-local-dns]** Adapt node-local-dns for air-gapped environments.
- **[node-manager]** deploy capi controller and webhooks before basic resources to prevent race condition during upgrades.
- **[registrypackages]** Added vex with CVE-2026-33186.

1.75.2

Know before update

- All Ingress-NGINX controller pods will be restarted.

Features

- **[cert-manager]** Add alerts for ACME Challenges stuck in pending or error states
- **[docs]** Add cloud providers dynamix, huaweicloud, and zvirt module architecture section.

Fixes

- **[ci]** Use GITHUB_TOKEN for backport action and set required workflow permissions
- **[cilium-hubble]** Fixed CVE-2026-33186 in the hubble-ui image.
- **[cni-cilium]** Fixed CVE-2026-33186, CVE-2026-27142, and CVE-2026-27139 by updating grpc dependency and Go version, and resolved build compatibility issues.
- **[common]** Fixed CVE-2026-24051 in the CoreDNS image.
- **[dhctl]** mitigate CVE-2026-33186
- **[ingress-nginx]** Nelm fixes are backported.
  All Ingress-NGINX controller pods will be restarted.

1.75.1

Know before update

- Registry pods will be restarted.
- ingressgateway, istio-cni-node, istiod, kiali, operator pods will be restarted after update

Features

- **[cloud-provider-zvirt]** add customNetworkConfig
- **[cloud-provider-zvirt]** add customNetworkConfig
- **[loki]** Added generation and usage of a dedicated TLS server certificate for Loki kube-rbac-proxy

Fixes

- **[admission-policy-engine]** Fix SecurityPolicyException handling for hostPorts-only exceptions
- **[cloud-provider-aws]** fix CVE in cloud-provider-aws
- **[cloud-provider-azure]** fix CVEs in cloud-provider-azure
- **[cloud-provider-dvp]** fix CVEs in cloud-provider-dvp
- **[cloud-provider-dvp]** fix CVEs in cloud-provider-dvp
- **[cloud-provider-dvp]** refactored CreateVolume to improve idempotency when disk.status.capacity is not yet reported and standardized gRPC error handling
- **[cloud-provider-huaweicloud]** fix CVEs in cloud-provider-huaweicloud
- **[cloud-provider-huaweicloud]** fix CVEs in cloud-provider-huaweicloud
- **[cloud-provider-openstack]** fix CVE in cloud-provider-openstack module
- **[cloud-provider-yandex]** fix CVEs in cloud-provider-yandex
- **[cloud-provider-zvirt]** fix CVEs in cloud-provider-zvirt

- **[cloud-provider-zvirt]** docs fox
- **[cni-cilium]** fixed mount-points in agent
- **[deckhouse]** Bump nelm version with deadlock fix.
- **[dhctl]** Fix restart bootstrap during creating additional nodes in cloud permanent node groups.
- **[dhctl]** Aix non-strict unmarshalling for metaconfigs.
- **[dhctl]** Fix deadlock in converge.
- **[docs]** Added the display of all used component versions in modules on the OSS info page.
- **[docs]** Add support for highlighting unknown code block types.
- **[docs]** Fix alerts rendering.
- **[docs]** Moving the platform update section to the top in the sidebar.
- **[docs]** Changing the term metka to label in Russian-language documentation.
- **[docs]** Escape HTML in OpenAPI descriptions v2
- **[docs]** Added info about using update-observer according to
- **[extended-monitoring]** fix typo in image-availability-exporter template
- **[istio]** cve fixes
  ingressgateway, istio-cni-node, istiod, kiali, operator pods will be restarted after update
- **[registry]** fixed path in images
- **[registry]** Updated auth image Go dependencies to fix Go CVEs.
  Registry pods will be restarted.
- **[registry-packages-proxy]** fix possible deadlock in cache retention policy
- **[tools]** Added HTTP client proxy support
- **[tools]** Bump lib-helm version to 1.71.2

Version 1.74

Important

- Support for Kubernetes 1.34 has been added, while support for Kubernetes 1.29 has been discontinued. In future DKP releases, support for Kubernetes 1.30 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.32.

Major changes

- Added support for managing experimental and alpha Kubernetes features (feature gates) using the `enabledFeatureGates` parameter of the `control-plane-manager` module. For details on managing these features in DKP, refer to the module documentation.

- The `--terminated-pod-gc-threshold` parameter (the termination threshold for evicted pods) is now calculated automatically based on the number of nodes in the cluster. This provides a balance between cluster stability and the ability to perform diagnostics. Threshold values are available in the `control-plane-manager` module documentation.

- The maximum number of pods (`maxPods`) for a node group is now set automatically based on the `podSubnetNodeCIDRPrefix` parameter from ClusterConfiguration (unless explicitly specified). This ensures stable cluster operation.

- Added the `vipAddress` parameter to the HuaweiCloudInstanceClass resource of the Huawei Cloud provider, allowing you to specify a virtual IP address for all nodes in the instance class.

- The Huawei Cloud provider configuration now allows overriding the primary network for the main network interface (`mainNetwork`) and specifying subnets for additional network interfaces (`additionalNetworks`) on both CloudPermanent and CloudEphemeral nodes.

  Warning. When upgrading to DKP 1.74 on Huawei Cloud, a bulk recreation of CloudEphemeral nodes may occur. Due to a temporary node shortage, some Pods may fail to start.

- The archive with debugging data is now collected using the `d8` tool. If you need to collect diagnostic data, follow the instructions.

- Now when hook execution fails, all metrics collected by that hook are preserved. This prevents loss of debugging information and simplifies the following root-cause analysis.

- Fixed a Deckhouse queue freeze in hybrid clusters that occurred when module configuration parameters were incomplete.

Security

- Implemented module integrity control based on the EROFS filesystem to improve security and prevent unauthorized modifications. The new system requires kernel support for dm-verity and EROFS.

Network

- Added support for the SCTP (Stream Control Transmission Protocol). Applications using this protocol can now run in a Deckhouse cluster.
- Added a `geoproxy` auxiliary microservice to the `ingress-nginx` module. This service is aimed at improving the stability of the Ingress NGINX Controller when working with GeoIP databases and provides the following features:
  - MaxMind license saving (databases are downloaded from a single point once a day).
  - Persistent data storage (if components are restarted, it doesn't require accessing the MaxMind servers).
  - Lets you specify a custom mirror for downloading databases.
- Added the Prometheus metric `bpf_progs_complexity_max_verified_insts` to assess the number of instructions in eBPF programs loaded into the kernel on cluster nodes. This metric helps evaluate the compatibility of the networking subsystem with the node kernels. The metric requires Linux kernel version 5.16 or newer.
- Added a new metric (`geoip_version`) and dashboards for monitoring GeoIP functionality in the cluster.
- Fixed issues with simultaneous updates of the `node-local-dns` module and the networking subsystem.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.74.18

Fixes

- **[deckhouse-controller]** Don't create an external module release for a module that is still shipped embedded, so it can't replace or duplicate the embedded copy.
- **[deckhouse-controller]** A module that conditionally depends on another is no longer disabled when an incompatible version of that dependency is enabled; the enable is rejected instead.
- **[istio]** Implement graceful metadata secret renewal for multiclusters.
- **[network-gateway]** Updated dnsmasq to v2.92-alt2 to address multiple security vulnerabilities (CVE-2026-*)
- **[node-local-dns]** Fix name of registry secret in safe-updater deployment

1.74.17

Fixes

- **[node-manager]** Added cleanup for oversized MCM MachineSet revision history annotation

1.74.16

Know before update

- All Ingress-NGINX controller pods will be restarted.
- All pods of Ingress-NGINX controller will be restarted.
- This update triggers a rolling update of the kube-proxy pods.
- istio module pods will be restarted
- module pods will be restarted

Features

- **[docs]** Image zooming and mobile view.
- **[docs]** Add sds-node-configurator module architecture section.
- **[docs]** Add cloud-provider-vcd module architecture section.

Fixes

- **[admission-policy-engine]** cve fixes for ratify
- **[cloud-provider-azure]** fix disk discovery for Ubuntu 22.04 Gen2 VMs with NVMe controllers
- **[cloud-provider-openstack]** Increase interval and timeout for health monitor
- **[cloud-provider-vsphere]** Fix vSphere privilege matrix and describe instructions for setting up environment via vSphere Client
- **[cni-simple-bridge]** Updated base image reference to support new Werf import format.
- **[common]** Fixed an issue where Trivy could not detect VEX files in coredns images.
- **[common]** Removed Python completely from the debug-container image as it is no longer needed, resolving corresponding CVEs, and silenced false positives for etcd binaries via VEX.
- **[deckhouse]** Remove notified=false annotation reset from runReleaseDeploy in the module release controller.
- **[deckhouse]** Fix webhook-handler CVEs.

- **[dhctl]** Added validation of the command execution status code
- **[docs]** Fix vSphere privilege matrix and describe instructions for setting up environment via vSphere Client
- **[documentation]** Fix search.
- **[ingress-nginx]** CVE-2025-15566 is backported.
  All pods of Ingress-NGINX controller will be restarted.
- **[ingress-nginx]** CVE-2026-4342 fix is backported to Dechkouse 1.74.
  All Ingress-NGINX controller pods will be restarted.
- **[istio]** fixed CVEs in module images
  module pods will be restarted
- **[istio]** fixed CVE-2026-39882, CVE-2026-39883 and CVE-2026-35206
  istio module pods will be restarted
- **[istio]** fixed CVE-2026-34986
  istio module pods will be restarted
- **[kube-proxy]** Fixed CVE-2026-33186 and CVE-2026-24051 in kube-proxy dependencies.
  This update triggers a rolling update of the kube-proxy pods.
- **[network-gateway]** Fixed werf import syntax for compatibility with older werf versions.

1.74.15

Know before update

- An additional service daemonset will be added.
- CoreDNS pods will undergo a rolling restart.
- pods in d8-istio namespace will be restarted

Features

- **[docs]** Add snapshot-controller module architecture section.

Fixes

- **[ci]** Use GITHUB_TOKEN for backport action and set required workflow permissions
- **[cilium-hubble]** Fixed CVE-2026-33186 in the hubble-ui image.
- **[cni-cilium]** Fixed CVE-2026-33186, CVE-2026-27142, and CVE-2026-27139 by updating grpc dependency and Go version, and resolved build compatibility issues.
- **[common]** Fixed CVE-2026-33186 in the CoreDNS image.
  CoreDNS pods will undergo a rolling restart.
- **[dhctl]** mitigate CVE-2026-33186
- **[istio]** fixed cve in module images
  pods in d8-istio namespace will be restarted
- **[node-local-dns]** Return stale-dns-connections-cleaner
  An additional service daemonset will be added.
- **[registrypackages]** Added vex with CVE-2026-33186.

1.74.14

Features

- **[docs]** Add cloud-provider-dvp module architecture section.
- **[docs]** Add local-path-provisioner module architecture section.
- **[docs]** Update architecture section.

Fixes

- **[cloud-provider-zvirt]** fix CSI token refresh patch apply
- **[docs]** Added the display of the d8 utility version on the command reference page.
- **[istio]** Reduce CPU and RAM for regenerate multicluster JWT token and sort ingressGateway
- **[tools]** Added HTTP client proxy support

1.74.13

Fixes

- **[cloud-provider-openstack]** fix LB.enabled flag

1.74.12

Know before update

- All Ingress-Nginx controller pods will be restarted.

Fixes

- **[candi]** Added a Netplan override to force the secondary NIC to use the main routing table, fixing cloud-init PBR conflicts.
- **[ingress-nginx]** CVE-2026-3288 fix is backported in all Ingress-Nginx controllers.
  All Ingress-Nginx controller pods will be restarted.
- **[node-manager]** Fixed vulnerabilities.

1.74.11

Know before update

- Kruise controller pod will be restarted.
- Registry pods will be restarted.

Fixes

- **[admission-policy-engine]** Fix template tests
- **[ci]** Add zVirt testing to e2e tests to DKP 1.74.
- **[ci]** Backport the provider instance change and the new containerd version in OpenStack tests to release 1.74.
- **[cloud-provider-aws]** fix getInstancesByIDs to comply with the describeInstanceBatcher.
- **[cloud-provider-openstack]** Add loadBalancer.enabled flag to prevent CCM crashes on k8s 1.32 without Octavia service
- **[docs]** Correct the wording in the cloud-provider-vsphere module docs.
- **[docs]** Updated editions for the modules managed-postgres, managed-valkey and managed-memcached.
- **[ingress-nginx]** Golang versions are updated.
  Kruise controller pod will be restarted.
- **[registry]** Updated auth image Go dependencies to fix Go CVEs.
  Registry pods will be restarted.

1.74.10

Know before update

- All ingress-nginx controller pods will be restared.

Fixes

- **[ingress-nginx]** Nginx and module’s dependencies are updated.
  All ingress-nginx controller pods will be restared.

1.74.9

Know before update

- Workload Pods are no longer denied by unrelated SecurityPolicy checks (e.g. hostNetwork/hostPort) when corresponding policy fields are not explicitly set.

Fixes

- **[admission-policy-engine]** Prevent unintended Gatekeeper constraints from being rendered for SecurityPolicy when boolean fields are omitted.
  Workload Pods are no longer denied by unrelated SecurityPolicy checks (e.g. hostNetwork/hostPort) when corresponding policy fields are not explicitly set.
- **[candi]** Bashible script fix to handle multiple GPUs
- **[candi]** fix if node has bashible-uninitialized taint in race condition.
- **[ci]** Add ignoring of D8NodeContainerdV2NotSupported and D8NodeCgroupV2NotSupported alerts to release 1.74.
- **[cloud-provider-dvp]** added fixes an infinite deletion state of DeckhouseMachine.
- **[deckhouse]** Fix yaml validation in settings.
- **[docs]** Changing module descriptions
- **[docs]** Editing FAQ problems applying dexprovider
- **[docs]** Editing flannel to cillium.
- **[docs]** Adding information about using images sign policy.
- **[istio]** Fixing the list of requests from istiod to gateway API
- **[registrypackages]** added usage GOPROXY in systemctl-power-commands-wrapper

1.74.8

Fixes

- **[docs]** Escape HTML in OpenAPI descriptions
- **[extended-monitoring]** Cleanup exporter metrics when the monitored resource has been

deleted

1.74.7

Fixes

- **[admission-policy-engine]** Convert constraints multi-line output to single-line
- **[candi]** Added a Netplan override to force the secondary NIC to use the main routing table, fixing cloud-init PBR conflicts.
- **[dhctl]** Fix panic in dhctl config render kubeadm-config command.
- **[istio]** Correction of an useless error in the Istio CNI workflow

1.74.6

Know before update

- All current connections powered by EgressGateways will be terminated.

Features

- **[docs]** Added info from `csi-ceph` module about Ceph storage system integration in DKP documentation.

Fixes

- **[cni-cilium]** Fix issue in generating CiliumEgressGatewayPolicy CR.
  All current connections powered by EgressGateways will be terminated.
- **[ingress-nginx]** A false-positive trigger of alert GeoIPDownloadErrorDetectedFromMaxMind is fixed.
- **[node-manager]** Fix panic in cluster-autoscaler caused by nil pointer dereference during node removal simulation.

1.74.5

Know before update

- The ingress nginx controllers’ pods will be restated.

Features

- **[candi]** Display lists of third-party (OSS) components in the modules documentation.
- **[cloud-provider-dvp]** Add validation for VirtualMachineClass and boot images before VM creation

Fixes

- **[admission-policy-engine]** Refactor constraint templates
- **[dhctl]** Improved reliability when connecting to dhctl servers by adding retry logic and better error handling during startup
- **[docs]** Corrections of typos and terms.
- **[docs]** Small imrovements of control-plane management docs
- **[docs]** Improve documentation for cluster network configuration
- **[ingress-nginx]** The CVE-2026-1580, CVE-2026-24512, CVE-2026-24513, CVE-2026-24514 CVEs fixes are backported.
  The ingress nginx controllers’ pods will be restated.
- **[ingress-nginx]** The CVE-2026-1580, CVE-2026-24512, CVE-2026-24513, CVE-2026-24514 CVEs are fixed.
  The ingress nginx controllers’ pods will be restated.
- **[monitoring-kubernetes]** fix kube_persistentvolume_is_local recording rule when there are more than one kube-state-metrics exporter in cluster
- **[prometheus]** Make Grafana redirect ingress pass the annotations validation.
- **[user-authz]** Updated logo on the authentication page.

1.74.4

Know before update

- If the SDN module is used in the cluster, the Cilium agent pods will be restarted.
- Improved documentation clarity for Yandex Cloud NAT instance deployment scenarios
- no impact

Features

- **[docs]** Add the enableNote display to the external modules configuration page.

- **[prometheus]** improve redirects from Grafana to the Deckhouse UI when Grafana is disabled
  no impact

Fixes

- **[cloud-provider-azure]** fixed patch in azure
- **[cloud-provider-dvp]** Fixed VM deletion timeout and improved memory validation error reporting
- **[cloud-provider-dvp]** Cleanup orphaned resources when VM creation fails
- **[cloud-provider-huaweicloud]** fix cve
- **[cloud-provider-vcd]** fix the interaction of legacy and non-legacy components with the API
- **[cloud-provider-vsphere]** fix cve
- **[cni-cilium]** Fix hook discovery_cni_exclusive.go
  If the SDN module is used in the cluster, the Cilium agent pods will be restarted.
- **[docs]** Fixed typo in the OpenStack integration section.
- **[docs]** Added Docker Bearer Token Realm configuration steps for Nexus registry setup.
- **[docs]** Document NAT instance deployment zones for different subnet configurations
  Improved documentation clarity for Yandex Cloud NAT instance deployment scenarios
- **[ingress-nginx]** Fixed useless geoip alerts.
- **[monitoring-kubernetes]** fix kube_persistentvolume_is_local recording rule for not-bound PVCs

1.74.3

Know before update

- Node-local-dns safe-uptader pods will be restarted.

Features

- **[prometheus]** Add prometheus configuration reload failed alert

Fixes

- **[admission-policy-engine]** Fix constraint templates, add additional tests
- **[cloud-provider-aws]** fix cve
- **[cloud-provider-vcd]** fix cve
- **[cloud-provider-yandex]** fix cve
- **[cni-cilium]** Fixed egress-gateway-agent controller logic for deleted resources and disable dev logging.
- **[deckhouse]** Fix module rerun.
- **[docs]** Added a link to the ModuleSource to the explanation table of the editions.
- **[metallb]** Fixed D8MetallbBGPSessionDown alert false positives caused by duplicate metrics without port information.
- **[node-local-dns]** The safe-updater’s logic of detecting if daemonset pods are up-to-date is improved.
  Node-local-dns safe-uptader pods will be restarted.
- **[registry]** Fixed validation of input image list changes in the registry checker.

1.74.2

Know before update

- All pods running kube-rbac-proxy will be restarted.

Features

- **[prometheus]** Replace PrometheusRules with ClusterObservabilityMetricsRulesGroups or ClusterObservabilityPropagatedMetricsRulesGroups when deployed using helm_lib_prometheus_rules helper and the observability module is enabled

Fixes

- **[admission-policy-engine]** fix namespaceSelector for constraints, add CRDs
- **[candi]** Remove duplicate `additional_disks_hashes` definition in static-node Terraform module.
- **[candi]** Updated the bashible step to include Linux kernel versions that address CVE-2025-37999
- **[candi]** Allow manually stopping DVP node VirtualMachines in nested clusters by using runPolicy AlwaysOnUnlessStoppedManually.
- **[cloud-provider-azure]** fixed cve
- **[cloud-provider-dvp]** this PR gives a lot more informational errors and messages to user
- **[cloud-provider-openstack]** fix cve
- **[common]** Latest CVEs are fixed.
  All pods running kube-rbac-proxy will be restarted.
- **[control-plane-manager]** Fix validation for resource-quota-overrides labels
- **[deckhouse]** Fix mpo cleanup.

- **[deckhouse]** Fix module installer cleanup.
- **[deckhouse-controller]** Exclude all service accounts from `d8-` namespaces in `d8ms-prefix` ValidatingAdmissionPolicy.
- **[ingress-nginx]** Latest CVEs are fixed.
  All pods running kube-rbac-proxy will be restarted.
- **[kube-dns]** Reverts
- **[log-shipper]** Fixed incorrect apiVersion in ClusterLogDestination documentation examples.
- **[node-manager]** Change NodeGroup maxPods validation from hard rejection to warning.
- **[registry]** Omitted the auth field in DockerConfig when credentials (username and password) are empty.

1.74.1

Know before update

- Prevents creation of Projects with too-long names that would lead to invalid generated Kubernetes resource names.
- deckhouse queue
- deckhouse queue blocked

Features

- **[common]** Enrich debug-container image with nsenter and /bin/sh symlink.
- **[common]** Resource quota ignore mechanism for pvc and pods
- **[dhctl]** Skipped application edition validation for standalone builds.

Fixes

- **[candi]** remove excessive netcat calls from d8-shutdown-inhibitor
- **[candi]** Add pause and kubernetes-api-proxy registry packages to bashible `bb-package-fetch` to prevent node failures during containerd major upgrades.
- **[control-plane-manager]** Fixed vulnerabilities
- **[deckhouse]** Fix module docs rendering.
- **[deckhouse]** Fix module enabling.
- **[deckhouse-controller]** fix copy-custom-certificate hook
- **[docs]** Added link to NVIDIA Container Toolkit docs.
- **[docs]** Fixed typo.
- **[docs]** Fixed the logic of generating the revision table.
- **[extended-monitoring]** Add namespace-scoped overrides
- **[ingress-nginx]** Improved stability of geoproxy service startup.
- **[kube-dns]** Refactored hook remove_old_coredns_deploy_and_svc — delegated deletion of original kube-dns service to helm.
- **[multitenancy-manager]** Add validation to restrict Project name length to 53 characters.
  Prevents creation of Projects with too-long names that would lead to invalid generated Kubernetes resource names.
- **[node-manager]** remove excessive netcat calls from d8-shutdown-inhibitor
- **[node-manager]** It fixes issues in the DaemonSet manifest for fencing module.
- **[tools]** Add check values not null in prometheusRule or ClusterObservabilityPropagatedMetricsRulesGroup
  deckhouse queue
- **[tools]** Add check values not null in prometheusRule or ClusterObservabilityPropagatedMetricsRulesGroup
  deckhouse queue blocked
- **[user-authn]** Quote service names to prevent digit-only names from breaking yaml parser
- **[user-authz]** Allow project-scoped roles to access Cluster-wide objects

Version 1.73

Important

- This release includes several important security improvements. Multiple known vulnerabilities have been fixed, including one in the user-authn module (CVE-2025-22868) that could potentially allow bypassing authentication checks. It is recommended that you schedule this update. See the Security section for details.
- The `dashboard` module will be removed in future DKP versions. Use the Deckhouse web UI instead (requires the `console` module to be enabled).
- The `runtime-audit-engine` module is now loaded from an external source (the `deckhouse` ModuleSource).
- All DKP components will be restarted during the update.

Major changes

- Added support for changing the registry settings (via the `mode` parameter) in `Unmanaged` mode, similar to `Direct` mode. A new parameter `checkMode` has been added to control registry validation behavior:
  - `Default`: Verifies the availability of all system module images and the deckhouse-controller images.
  - `Relax`: Verifies only the deckhouse-controller images.
- Updated requirements for the `email` field of the User object: the email address must now be in lowercase. Existing users will not be affected.
- Dex updated to **v2.44.0**. It now allows authentication through available identity providers if one of them is down, and supports authentication via identity providers through a proxy.
- The User object status now displays the reason for user lockout (controlled by the `lockout` parameter).
- Added the `additionalDisks` parameter for the Deckhouse Virtualization Platform integration provider, allowing creation and attachment of additional disks to VMs in a NodeGroup (`size` and StorageClass must be specified). This simplifies data distribution across multiple disks.
- Added support for `additionalMetadata` in objects (networks, VMs, disks) for the VMware Cloud Director integration provider. Metadata is merged with existing data, with `additionalMetadata` values taking precedence. Changing `additionalMetadata` triggers recreation of CloudEphemeral node groups that use the affected VCDInstanceClass.
- For the VMware vSphere integration provider, you can now specify an SPBM storage policy ID (via the `storagePolicyID` parameter) and configure automatic creation of a StorageClass for each available SPBM policy. You can now explicitly select a policy for master and worker nodes and use the corresponding storage classes.
- Added alerts to help plan module deprecation or migration:
  - `ModuleIsDeprecated`: Notifies when a module is deprecated and nearing end of support.
  - `D8ModuleOutdatedByMajorVersion`: Notifies when a module is behind by one or more major versions.
- Added `GeoIPDownloadErrorDetected` alert to notify about MaxMind GeoIP database download issues.
- The update notification workflow has changed — a release is applied only after the notification is successfully delivered to the configured webhook. If delivery fails, the update is paused until the webhook is restored.
- Reorganized in-cluster documentation. All module documentation (including connected ones) is now located under the Modules section. Search has been updated.
- For Ingress NGINX Controller v1.10, added the option to enable the profiler (via the `nginxProfilingEnabled` parameter). Enabling the profiler increases resource consumption but may be useful for debugging controller issues.
- Added support for custom HTTP authentication headers (via the `headers` parameter of UpmeterRemoteWrite) when sending SLA monitoring metrics via Prometheus Remote Write protocol.
- Optimized DKP core module loading, reducing startup time in clusters with many modules.
- Audit logs now show which OIDC provider issued the authentication token
- Deckhouse CLI (`d8`) updated to v0.23.0:
  - Added the `d8 status` command, which provides a quick cluster summary (nodes, releases, Deckhouse pods, alerts, registry, Deckhouse settings, CNI, queue state).
  - Added the `d8 k debug` command, which runs the DKP built-in debug container (image can be overridden via `--image`). This simplifies interactive pod debugging.
  - Added the `--watch` flag to the `d8 system queue list` command to track queue state changes.

Security

- Improved baseline container security. Updated security profiles for the following modules to restrict privileges and access rights to the minimum required: `cni-cilium`, `cni-flannel`, `cni-simple-bridge`, `ingress-nginx`, `istio`, `keepalived`, `kube-dns`, `kube-proxy`, `node-local-dns`, `network-gateway`, `network-policy-engine`, `open-vpn`.
- Added the `allowRbacWildcards` flag to the SecurityPolicy, controlling whether wildcards are

allowed in Role and RoleBinding definitions (set to `true` by default). Security policies can now also restrict interactive connections to pods (`CONNECT` for `pods/exec` and `pods/attach`) within namespaces.

- Added support for preventing creation of pods with specific tolerations from a list (`policies.disallowedTolerations` parameter in the operational policy). This helps prevent user workloads from running on nodes reserved for special tasks.
- Enhanced security in Ingress NGINX Controller v1.12 (distroless image, vulnerability fixes, and other improvements).
- Fixed known vulnerabilities in the following modules: `operator-trivy`, `registry`, `user-authn`, `cloud-provider-dvp`, `multitenancy-manager`, `admission-policy-engine`, `ingress-nginx`, `alertmanager`, `metallb`, `istio`, `node-local-dns`, `kube-apiserver`.

Network

- For the VMware Cloud Director integration provider, added support for LoadBalancer backed by VMware NSX Advanced Load Balancer (ALB/Avi) when using `NSX-T`. Requires the Load Balancer feature to be enabled on the Edge Gateway. If LoadBalancer is enabled after cluster creation, DKP automatically detects and applies the change within an hour. For open ports, DKP creates corresponding `Pool + Virtual Service` pairs. If there is a firewall, add allow rules for the LoadBalancer's external IP address and relevant ports.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.73.14

Know before update

- All Ingress-Nginx controller pods will be restarted.
- All ingress-nginx controller pods will be restared.
- All modules of the ingress-nginx module will be restarted.
- Registry pods will be restarted.
- The ingress nginx controllers' pods will be restated.

Fixes

- **[admission-policy-engine]** added mount-points in ratify
- **[admission-policy-engine]** deny_vulnerable_images webhook must use module status instead of moduleconfig
- **[candi]** cve fix for admission-policy-engine, cert-manager, user-authn, multitenancy-manager
- **[candi]** Updated pwru tool to v1.0.11 to fix CVE-2025-68121.
- **[candi]** Updated pwru tool to v1.0.11 to fix CVE-2025-68121.
- **[chrony]** mitigate CVE-2025-58181
- **[common]** Fixed vulnerability CVE-2026-24051.
- **[control-plane-manager]** Fixed vulnerabilities.
- **[documentation]** Fixed HTTP/404 error when opening modules documentation from the Web interface.
- **[extended-monitoring]** Fixed CVE-2025-47914, CVE-2025-58181
- **[ingress-nginx]** CVE-2026-3288 fix is backported in all Ingress-Nginx controllers.
  All Ingress-Nginx controller pods will be restarted.
- **[ingress-nginx]** Golang and Altlinux updates are reverted.
  All modules of the ingress-nginx module will be restarted.
- **[ingress-nginx]** Nginx and module's dependencies are updated.
  All ingress-nginx controller pods will be restared.
- **[ingress-nginx]** The CVE-2026-1580, CVE-2026-24512, CVE-2026-24513, CVE-2026-24514 CVEs fixes are backported.
  The ingress nginx controllers' pods will be restated.
- **[loki]** Fixed CVE-2025-47914, CVE-2025-58181
- **[monitoring-kubernetes]** Fixed CVE-2025-47914, CVE-2025-58181
- **[node-local-dns]** Fixed CVEs
- **[node-manager]** Fixed vulnerabilities.
- **[operator-prometheus]** Fixed CVE-2025-47914, CVE-2025-58181
- **[prometheus]** Fixed CVE-2025-47914, CVE-2025-58181, CVE-2025-65637
- **[prometheus-metrics-adapter]** Fixed CVE-2025-47914, CVE-2025-58181
- **[prometheus-pushgateway]** Fixed CVE-2025-47914, CVE-2025-58181, CVE-2025-22872, CVE-2025-22868
- **[registry]** Updated auth image Go dependencies to fix Go CVEs.
  Registry pods will be restarted.
- **[registry]** Updated auth image Go dependencies to fix Go CVEs.
  Registry pods will be restarted.

- **[registrypackages]** Fixed vulnerability CVE-2026-24051.
- **[upmeter]** Add proper securityContext to the upmeter probe to meet the restricted security profile constraints.
- **[upmeter]** Fixed CVE-2025-47914, CVE-2025-58181, CVE-2025-65637

1.73.13

Know before update

- config module loki ↓

Fixes

- **[candi]** Updated the bashible step to include Linux kernel versions that address CVE-2025-37999
- **[cni-cilium]** Fixed vulnerabilities.
- **[cni-flannel]** Bump version and fixed vulnerabilities.
- **[istio]** Fix CVE for Istio version 1.21 and 1.25
- **[kube-dns]** Fixed vulnerabilities.
- **[kube-proxy]** Fixed vulnerabilities.
- **[kube-proxy]** Fixed vulnerabilities.
- **[loki]** disable send analytics report to stats.grafana.org
  config module loki ↓
- **[network-policy-engine]** Fixed vulnerabilities.
- **[node-local-dns]** Bump version and fixed vulnerabilities.
- **[openvpn]** Fixed vulnerabilities.
- **[registry]** Omitted the auth field in DockerConfig when credentials (username and password) are empty.

1.73.12

Fixes

- **[cloud-provider-vsphere]** fix stale session for cloud-data-discoverer
- **[istio]** fixing the CVE in Kiali

1.73.11

Fixes

- **[dashboard]** Fixed CVE-2025-30204 by updating dashboard components
- **[deckhouse]** Fix module enabling.
- **[registrypackages]** Update integrity patch for containerd (cse only).

1.73.10

Fixes

- **[cloud-provider-dvp]** Added functionality to wait for a disk to be attached to a VM
- **[cloud-provider-huaweicloud]** fix CSI unpublishValidation for non exist ECS instance
- **[registrypackages]** Update integrity patch for containerd (cse only).
- **[service-with-healthchecks]** Fixed CVEs
- **[user-authz]** cache namespace label checks in the user-authz webhook via informer to avoid per-request apiserver GETs

1.73.9

Features

- **[ci]** Add DVP e2e for the cse edition

Fixes

- **[deckhouse-controller]** Fix conversions for external modules
- **[dhctl]** Fix parallel bootstrap cloud permanent nodes

1.73.8

Know before update

- Ingress Nginx controller dashboards are fixed
- The MTU will be updated on all interfaces of all pods.

Fixes

- **[cloud-provider-yandex]** cloud-provider-yandex CVE’s was fixed
- **[cni-cilium]** The MTU configuration has been updated.
  The MTU will be updated on all interfaces of all pods.
- **[deckhouse-controller]** Fix “multiple readiness hooks found” error on hook registration retry after failure.
- **[deckhouse-controller]** Fix deprecated and experimental module metrics not being reset when module is disabled.
- **[dhctl]** Fix panic during destroy. Change opentofu log level to INFO.
- **[docs]** The current editions are listed on the page about installation in a closed environment.
- **[docs]** Fixed links to the Getting Started in the documentation.
- **[prometheus]** Fix namespace label value in the Ingress Nginx controller and several other metrics
  Ingress Nginx controller dashboards are fixed
- **[user-authn]** Fix BadRequest after the change password redirect when password policy is enabled
- **[user-authn]** Fix login error 500 with password policy enabled.

1.73.7

Know before update

- Allows scaling control-plane from 1→3 in clusters where ControlPlaneKubeletLocalMode=true.
- The instructions for reducing the number of master nodes have not been updated for a long time, and for a long time now, the conversion to reducing the number of master nodes in clusters in a cloud installation is automatic.
- upmeter check

Features

- **[ci]** Add Deckhouse Virtualization Platform testing to e2e tests.
- **[docs]** Added Commander support for DVP EE in the Platform Editions table.

Fixes

- **[candi]** Add missing debian 13 version to detect_bundle
- **[cloud-provider-huaweicloud]** fix Provider ID
- **[control-plane-manager]** Fix “etcd join” phase for control-plane scaling in v1.33.
  Allows scaling control-plane from 1→3 in clusters where ControlPlaneKubeletLocalMode=true.
- **[deckhouse-controller]** Fixed verifying migrated modules
- **[deckhouse-controller]** fix conversion applying for external modules
- **[dhctl]** Fix StaticInstance readiness check and refactoring readiness check for resources.
- **[docs]** Outdated instructions for reducing the number of master nodes in cloud clusters have been fixed. Users will correctly perform the converge of cloud clusters.
  The instructions for reducing the number of master nodes have not been updated for a long time, and for a long time now, the conversion to reducing the number of master nodes in clusters in a cloud installation is automatic.
- **[kube-dns]** Improved /etc/hosts renderer compatibility with admission-policy-engine Restricted mode.
- **[log-shipper]** Update documentation.
- **[upmeter]** fix securityxontext for statefulset
  upmeter check

1.73.6

Fixes

- **[cloud-provider-vsphere]** fix cloud-data-discoverer (SPBM)

1.73.5

Know before update

- node-exporter pods will be rollout restarted during upgrade

Features

- **[ci]** Add static e2e for the cse edition
- **[ci]** Add static e2e for the cse edition
- **[docs]** Added documentation about configuration of the Kubernetes audit policy.

Fixes

- **[cloud-provider-azure]** fix build image for azure ccm
- **[cni-cilium]** Some issues have been fixed in the EgressGateway.

- **[dhctl]** Fix converge manifests for static cluster in commander.
- **[monitoring-kubernetes]** remove the Docker traces from the module code
  node-exporter pods will be rollout restarted during upgrade

1.73.4

Know before update

- Ingress-Nginx controller’s pods of 1.9 version will be restarted.
- erroneous alerts for node disk usage are fixed

Fixes

- **[cloud-provider-dvp]** add missing field to Cluster
- **[cloud-provider-yandex]** Terraform auto converger was failed for WithNATInstance layout.
- **[deckhouse]** Fix validation logic for a disabled module
- **[dhctl]** Validate WithNATInstance Yandex layout params only in bootstrap.
- **[dhctl]** Added nil check to dhctl during converge in migrator
- **[extended-monitoring]** drop metrics when extended monitoring is disabled for node(s)
  erroneous alerts for node disk usage are fixed
- **[ingress-nginx]** A symlink to the new opentelemetry config path is added.
  Ingress-Nginx controller’s pods of 1.9 version will be restarted.
- **[ingress-nginx]** Fixed CVEs
- **[monitoring-kubernetes]** Rollout changes for resources metrics kubelet
- **[node-manager]** Fix panic in registry packages proxy if image not found.
- **[operator-trivy]** Fix CIS Benchmark report template
- **[operator-trivy]** fix CVE-2025-52881 CVE-2024-25621 on image trivy
- **[registrypackages]** Update containerd to 1.7.29 / 2.1.5 and runc to 1.3.3

1.73.3

Know before update

- the extended monitoring will only be enabled when the label is explicitly set on a namespace

Fixes

- **[control-plane-manager]** Add vex for CVE-2025-31133, CVE-2025-52881 .
- **[deckhouse]** Add umask before creating first-control-plane-bashible-ran file
- **[dhctl]** Fix getting passphrase for key from connection config for cli.
- **[dhctl]** Move yandex withNATInstance layout settings from preflights to preparator.
- **[documentation]** Fix module’s deployment and configuration.
- **[extended-monitoring]** Fix extended-monitoring.deckhouse.io/enabled label handling
  the extended monitoring will only be enabled when the label is explicitly set on a namespace
- **[monitoring-kubernetes]** fix CVE-2025-52881 for node-exporter
- **[prometheus]** Fix description for not usable CVE
- **[registrypackages]** Fixes CVE in kubernetes-cni

1.73.2

Fixes

- **[ingress-nginx]** CVEs fixed
- **[istio]** Kiali’s CVE-2024-28180 is fixed.
- **[multitenancy-manager]** fix CVE-2024-25621 CVE-2025-64329
- **[node-manager]** fix cve and vex
- **[operator-trivy]** fix CVE-2025-52881 CVE-2024-25621
- **[operator-trivy]** Add grep to node-collector and improve error reporting
- **[user-authn]** Rollback patch for handling insecureSkipEmailVerified condition

1.73.1

Features

- **[docs]** Added manifest for internal LB VK Cloud.
- **[docs]** Added info from `csi-netapp` module about NetApp storage integration in DKP documentation.
- **[istio]** Added access log format setting for proxy sidecars.

Fixes

- **[admission-policy-engine]** Prohibit only creation or modification for objects with vulnerable images

- **[admission-policy-engine]** Fixed proxy support for trivy-provider
- **[candi]** Remove proxy-environment for kubelet when proxy disabled
- **[ci]** Fix e2e upgrade tests.
- **[ci]** fix grafana vex
- **[ci]** fix trickster vex
- **[ci]** Fix missing edition suffixes in prerelease build.
- **[cloud-provider-dvp]** Correct the calculation of the path to the device
- **[cloud-provider-vsphere]** fix vSphere storageClass template
- **[cni-cilium]** fixed connectivity dashboard
- **[deckhouse]** Fix race conditions in test
- **[deckhouse-controller]** fix bash completion
- **[docs]** Small fix on the “Security event audit” page.
- **[docs]** Fixed the description for the 4239 port in the reference table.
- **[docs]** Updated installer block in GS.
- **[docs]** Added a link to the DVP integration page in the virtualization overview.
- **[istio]** The same owner is specified for the files that are used to run in the operator container.
- **[metallb]** Disabled unnecessary speaker functionality.
- **[monitoring-kubernetes]** Fixed CVE-2025-22870, CVE-2025-22872.
- **[prometheus]** Fix plur in vex for tricster
- **[prometheus-pushgateway]** Fixed CVE-2025-22869, CVE-2025-22868, CVE-2025-22870.
- **[registrypackages]** Update runc to 1.3.1.

Version 1.72

Important

- All DKP components will be restarted during the update.
- To use experimental modules in the cluster, you now need to explicitly enable the allowExperimentalModules parameter. By default, experimental modules are disabled. Modules that were enabled before the update will not be automatically disabled. However, if an experimental module enabled prior to the update is manually disabled during the update process, you will need to grant permission to use experimental modules again in order to re-enable it.
- If there are WireGuard interfaces on the cluster nodes, you must update the Linux kernel to version 6.8 or higher.

Major changes

- Added a new registry module and the ability to adjust container registry parameters without restarting all DKP components. Two modes for working with the container registry are now available in DKP: `Unmanaged` (the approach used in previous versions) and `Direct` (a new mode). In `Direct` mode, DKP creates a virtual container registry address in the cluster that all DKP components use. Changing the container registry address (for example, switching to a different registry or changing the DKP edition in the cluster) in this mode does not trigger a forced restart of all DKP components.

- Added support for recursive DNS servers (configured via the recursiveSettings section of the `cert-manager` module). They are used to verify the existence of a DNS record before starting the ACME DNS-01 domain ownership validation process. This is useful if the same domain is used both publicly and within the cluster, or if the domain has dedicated authoritative DNS servers.

- Introduced separation of modules into critical and functional using the `critical` flag in `module.yaml`. Critical modules are started first. Functional modules are started after the bootstrap process is complete. Their tasks run in parallel and do not block the queue in case of failure. This speeds up cluster installation and improves fault tolerance when starting modules.

- You can now enable logging of all DNS queries (the enableLogs parameter of the `node-local-dns` module).

- In the `cloud-provider-vcd` module, a new WithNAT layout has been added for cluster deployment. It automatically configures NAT and, if necessary, firewall rules for accessing nodes through a bastion host. It also supports both `NSX-T` and `NSX-V`. This makes it possible to deploy a cluster in VMware Cloud Director without pre-configuring the environment (unlike the `Standard` layout).

Security

- Added the fields `user-authn.deckhouse.io/name` and `user-authn.deckhouse.io/preferred_username` to Kubernetes audit log events. These fields display user claims from the OIDC provider, improving authentication monitoring and troubleshooting.

- Kubernetes versions 1.30–1.33 have been updated to the latest patch releases.

- For the AWS provider, added the ability to disable the creation of default security groups (the disableDefaultSecurityGroup parameter). When disabled, security groups must be created manually and explicitly specified in AWSClusterConfiguration, AWSInstanceClass, and NodeGroup. This new feature provides greater control over security settings.
- Added support for password policies for local users (configured in the passwordPolicy section). You can now enforce a minimum password complexity, set password expiration, require password rotation, prevent reuse of old passwords, and lock accounts after a specified number of failed login attempts. These changes allow administrators to centrally enforce password requirements and improve cluster security.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.72.13

Fixes

- **[istio]** Reduce CPU and RAM for regenerate multicluster JWT token and sort ingressGateway

1.72.12

Fixes

- **[deckhouse-controller]** Fix conversions for external modules
- **[deckhouse-controller]** Fixed verifying migrated modules
- **[deckhouse-controller]** Fixed a crash during external module updates with conversions that caused ModuleRelease to fail validation due to a forbidden property error.
- **[istio]** Fixed AuthorizationPolicy CRD insufficiency for Istio 1.25.

1.72.11

Fixes

- **[cert-manager]** Mark module as critical for cluster
- **[deckhouse-controller]** fix deckhouse update accidentally minor skip
- **[istio]** Erroneous option in 1.25 control-plane helm template fixed.
- **[registrypackages]** Fixed permissions for directory with cni-plugins in PCI-DSS clusters
- **[tools]** update dmt version to 0.1.53

1.72.10

Fixes

- **[ci]** Delete an extra entry from the changelog.
- **[cloud-provider-vsphere]** fix hook logick
- **[deckhouse-controller]** fix possibility deadlock
- **[istio]** Reduce RAM for regenerate multicluster JWT token

1.72.9

Fixes

- **[candi]** Switch node hostname in discover_node_ip.sh to locked hostname
- **[candi]** fixed version of containerd image in bashible step
- **[deckhouse]** Fixes bug when module`s config values unchanged after enabling.
- **[deckhouse-controller]** Add more mutexes to shell-operator framework
- **[dhctl]** Change default ssh mode
- **[ingress-nginx]** The is reverted.
- **[ingress-nginx]** Enable ExecuteHookOnEvents on hook set_annotation_validation_suspended.go

1.72.8

Fixes

- **[cloud-provider-dvp]** fix single-node LoadBalancer bug and add multi-LB per node support
- **[cloud-provider-vsphere]** cloud-data-discoverer fixes
- **[node-manager]** Enable SSHLegacyMode if SSHPrivateKey is not empty and fix gossh panic.

1.72.7

Fixes

- **[cloud-provider-huaweicloud]** add rule to d8:cloud-provider-huaweicloud:cloud-controller-manager account for list pods as needed
- **[node-manager]** CAPI crd served version fix
- **[node-manager]** shutdown inhibitor use three-state GracefulShutdownPostpone condition so kubelet waits for the final decision before ending pods

1.72.6

Fixes

- **[candi]** Fix nodeuser deletion.
- **[cloud-provider-vsphere]** cloud-data-discoverer fixes
- **[deckhouse-controller]** Ensure `afterDeleteHelm` hooks receive Kubernetes snapshots by stopping monitors after hook execution
- **[dhctl]** Fixing bashible steps when it does not respond to the limit on the number of attempts
- **[dhctl]** Fix start and keep-alive behavior, do not change host if any kube-proxies has been started.
- **[node-manager]** CAPI crd served version fix
- **[vertical-pod-autoscaler]** Disable vpa for k8s-metacollector in module runtime-audit-engine

1.72.5

Fixes

- **[cloud-provider-vsphere]** cloud-data-discoverer fixes
- **[cni-cilium]** Fixed 1.14 -> 1.17 migration mechanism
- **[cni-cilium]** improved migration to 1.17 logic
- **[node-manager]** capi crd served version fix

1.72.4

Fixes

- **[cloud-provider-openstack]** fix openstack ccm deployment
- **[deckhouse]** Fix maintenance mode label ignore rule.
- **[ingress-nginx]** lowers the severity level of the IngressNginxController alert from 3 to 4

1.72.3

Fixes

- **[dhctl]** Add timeout to ssh client dial and fix stop.
- **[dhctl]** Fix ssh client start with bastion in dhctl.
- **[dhctl]** Fix deadlock while reading dhctl singlethreaded server logs
- **[dhctl]** Fixed the behaviour of the attach operation when there are no keys in SSHConfig, only a password
- **[docs]** Containerdv2 migration FAQ fix
- **[ingress-nginx]** fix set_annotation_validation_suspended.go hook to set correct metric
- **[ingress-nginx]** fix set_annotation_validation_suspended.go hook to set correct metric
- **[loki]** Change stage to Preview

1.72.2

Know before update

- The ingress-nginx pods of 1.10 will be restarted.

Fixes

- **[docs]** Added the Local path provisioner storage section to the documentation sidebar, including Overview, Examples, Reference, and FAQ entries.
- **[ingress-nginx]** Fixed nginx segfaults when opentelemetry is enabled.
  The ingress-nginx pods of 1.10 will be restarted.
- **[node-manager]** Fix node update approval hook for CAPI nodes.

1.72.1

Fixes

- **[deckhouse]** add check for user-authn in enabled script
- **[deckhouse]** fix editions list in check deckhouse edition hook
- **[dhctl]** Fix ssh client initialising in Commander Attach and Commander Detach operations

- **[node-manager]** Some minor fixes for nvidia gpu templates to improve security.

Version 1.71

Important

- Prometheus has been replaced with Deckhouse Prom++. If you want to keep using Prometheus, disable the `prompp` module manually before upgrading DKP by running the command `d8 system module disable prompp`.
- Support for Kubernetes 1.33 has been added, while support for Kubernetes 1.28 has been discontinued. In future DKP releases, support for Kubernetes 1.29 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.31.
- Upgrading the cluster to Kubernetes 1.31 requires a sequential update of all nodes, with each node drained. You can control how node updates requiring workload disruptions are applied using the `disruptions` parameter section.
- The built-in `snapshot-controller` and `static-routing-manager` modules will now be replaced with their external counterparts of the same name, sourced via ModuleSource `deckhouse`.
- The new version of Cilium requires nodes to run Linux kernel version 5.8 or newer. If any node in the cluster has a kernel older than 5.8, the Deckhouse Kubernetes Platform upgrade will be blocked. Cilium Pods will be restarted.
- All DKP components will be restarted during the update.

Major changes

- You can now enforce two-factor authentication for static users. This is configured via the `staticUsers2FA` parameter section of the `user-authn` module.
- Added support for GPUs on nodes. Three GPU resource sharing modes are now available: Exclusive (no sharing), TimeSlicing (time-based sharing), and MIG (a single GPU split into multiple instances). The NodeGroup spec.gpu parameter section is used to configure the GPU resource sharing mode. Using a GPU on a node requires installing the NVIDIA Container Toolkit and the GPU driver.
- When enabling a module (with `d8 system module enable`) or editing a ModuleConfig resource, a warning is now displayed if multiple module sources are found. In such a case, explicitly specify the module source using the source parameter in the module's configuration.
- Improved error handling for module configuration. Module-related errors no longer block DKP operations. Instead, they are now displayed in the status fields of Module and ModuleRelease objects.
- Improved virtualization support:
  - Added a provider for integration with Deckhouse Virtualization Platform (DVP), enabling deployment of DKP clusters on top of DVP.
  - Added support for nested virtualization on nodes in the `cni-cilium` module.
- The `node-manager` module now includes several enhancements for improved node reliability and manageability:
  - You can now prevent a node from restarting if it still hosts critical Pods (labeled with `pod.deckhouse.io/inhibit-node-shutdown`). This can be necessary for workloads with stateful components, such as long-running data migrations.
  - Introduced API version `v1alpha2` for the SSHCredential resource, where the `sudoPasswordEncoded` parameter allows specifying the `sudo` password in Base64 format.
  - The `capiEmergencyBrake` parameter allows you to disable Cluster API (CAPI) in emergency scenarios, preventing potentially destructive changes. Its behavior is similar to the existing `mcmEmergencyBrake` setting.
- Added a pre-installation check to verify connectivity to the DKP container image registry.
- Improved the log file rotation mechanism when using short-term log storage (via the `loki` module). Added the `LokiInsufficientDiskForRetention` alert to warn about insufficient disk space for log retention.
- The documentation now includes a reference for the Deckhouse CLI (`d8` utility) commands and parameters.
- When using CEF encoding for collecting logs from Apache Kafka or socket sources, you can now configure auxiliary CEF fields such as Device Product, Device Vendor, and Device ID.

- The `passwordHash` field in the NodeUser resource is no longer required. This allows you to create users without passwords — for example, in clusters that use external authentication systems (such as PAM or LDAP).
- Added support for CRI Containerd v2 with CgroupsV2. The new version introduces a different configuration format and includes a mechanism to migrate between Containerd v1 and v2. You can change the CRI type used on nodes via the `cri.type` parameter and configure it using `cri.containerdV2`.

Security

- Container image signature verification is now available in DKP SE+. This feature is now supported in DKP SE+ and EE.
- The `log-shipper`, `deckhouse-controller`, and `Istio` (version 1.21) modules have been migrated to distroless builds. This improves security and ensures a more transparent and controlled build process.
- New audit rules have been added to track interactions with containerd. The following are now monitored: access to the `/run/containerd/containerd.sock` socket, modifications to the `/etc/containerd` and `/var/lib/containerd` directories and the `/opt/deckhouse/bin/containerd` file.
- Known vulnerabilities have been fixed in the following modules: `loki`, `extended-monitoring`, `operator-prometheus`, `prometheus`, `prometheus-metrics-adapter`, `user-authn`, and `cloud-provider-zvirt`.

Network

- Added support for Istio version 1.25.2, which uses the Sail operator instead of the deprecated Istio Operator. Also added support for Kiali version 2.7, without Ambient Mesh support. Istio version 1.19 is now considered deprecated.
- Added support for encrypting traffic between nodes and Pods using the WireGuard protocol (via the `encryption.mode` parameter).
- Fixed the logic for determining service readiness in the ServiceWithHealthcheck resource. Previously, Pods without an IP address (for example, in `Pending` state) could be mistakenly included in the load balancing list.
- Added support for the least-conn load balancing algorithm. This algorithm directs traffic to the service backend with the fewest active connections, improving performance for connection-heavy applications (such as WebSocket services). To use this algorithm, enable the `extraLoadBalancerAlgorithmsEnabled` parameter in the `cni-cilium` module settings and use the `service.cilium.io/lb-algorithm` annotation on the service and set it to a supported value: random, maglev, or least-conn.
- Fixed an issue in Cilium 1.17 `cilium-operator` where IP addresses were not reused after a `CiliumEndpoint` was deleted. The issue was caused by improper cleanup of priority filters, which could lead to IP pool exhaustion in large clusters.
- Refined the list of ports used for networking:
  - Added and updated:
    - `4287/UDP`: WireGuard port used for CNI Cilium traffic encryption.
    - `4295-4297/UDP`: Used by the `cni-cilium` module for VXLAN encapsulation of inter-pod traffic in multiple nested virtualization — when DKP with the `virtualization` module enabled is deployed inside virtual machines that are also created in DKP with the `virtualization` module enabled.
    - `4298/UDP`: Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods if the cluster was deployed on DKP version starting from **1.71** (for clusters deployed on DKP versions up to **1.71**, see the note for ports `4299/UDP`, `8469/UDP`, and `8472/UDP`).
    - `4299/UDP`: Port **for clusters deployed on DKP versions 1.64–1.70.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the port used unless the `virtualization` module is enabled.
    - `8469/UDP`: Port **for clusters deployed on DKP version 1.63 and below with the `virtualization` module enabled prior to DKP version 1.63.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port
    - `8472/UDP`: Port **for clusters deployed on DKP version 1.63 and below.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port if the `virtualization` module

is not enabled. **Note** that in such clusters, enabling the `virtualization` module on DKP before version 1.70 changes the port:
  - Enabling the `virtualization` module on DKP version 1.63 and below will change it to `8469/UDP` and will not change with subsequent DKP updates
  - Enabling the `virtualization` module on DKP starting from version 1.64 will change it to `4298/UDP` and will not change with subsequent DKP updates
- Removed:
  - `49152`, `49153/TCP`: Previously used for live migration of virtual machines (in the `virtualization` module). Migration now occurs over the Pod network.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.71.10

Features

- **[deckhouse-controller]** ignore migrate module if disabled

Fixes

- **[deckhouse-controller]** fix deckhouse update accidentally minor skip
- **[istio]** Reduce CPU and RAM for regenerate multicluster JWT token and sort ingressGateway
- **[local-path-provisioner]** Fix CVE in local-path-provisioner module

1.71.9

Fixes

- **[cni-cilium]** Fixed 1.14 -> 1.17 migration mechanism

1.71.8

Fixes

- **[deckhouse]** Fix module source updating fails.
- **[vertical-pod-autoscaler]** Disable vpa for k8s-metacollector

1.71.7

Fixes

- **[candi]** Fix arg `encryption-provider-config` for kubeadm configuration.
- **[candi]** Disable immutable flag on erofs files in cleanup node stage.

1.71.6

Fixes

- **[cni-cilium]** Fixed the infinite loop in the “cilium migration” bashible step and improved synchronization between bashible and safe-agent-updater.

1.71.5

Know before update

- If wireguard interface is present on nodes, then cilium-agent upgrade will stuck. Upgrading the linux kernel to 6.8 is required.
- main and longterm Prometheuses will be rollout-restarted

Fixes

- **[candi]** Changed ExecStartPre in d8-shutdown-inhibitor.service
- **[candi]** Added warnings to the VMware Cloud Director environment documentation about Edge requirements
- **[cni-cilium]** Add a compatibility check for the Cilium version and the kernel version, if WireGuard is installed on the node
  If wireguard interface is present on nodes, then cilium-agent upgrade will stuck. Upgrading the linux kernel to 6.8 is required.
- **[cni-cilium]** Added a migration mechanism, which was implemented through the node group disruptive updates with approval.
- **[cni-cilium]** fixed invalid annotation name for lb-algorithm in docs
- **[deckhouse-controller]** fix module config ensure

- **[docs]** Fix relative links in the multitenancy-manager module documentation.
- **[docs]** Added steps that patch secret and prevented the image pull fail.
- **[docs]** Add containerv2 additional registry examples
- **[docs]** Add new requirement and commands to meet containerdv2
- **[docs]** Fixed command syntax for Docker container run in documentation.
- **[docs]** Add new requirement and commands to meet containerdv2 requirements
- **[docs]** Fix D8KubernetesStaleTokensDetected alert description.
- **[docs]** There should be one disk in the template.
- **[docs]** Updates for Observability docs.
- **[log-shipper]** Documentation updates.
- **[loki]** Documentation updates.
- **[monitoring-custom]** Documentation updates.
- **[monitoring-kubernetes]** Documentation updates.
- **[monitoring-ping]** Documentation updates.
- **[prometheus]** fix securityContext indentation in the Prometheus main and longterm resources main and longterm Prometheuses will be rollout-restarted
- **[prometheus]** Documentation updates.
- **[upmeter]** Documentation updates.

1.71.4

Know before update

- low

Features

- **[docs]** Added descriptions of the problem of using Cilium and Elastic together
  low

Fixes

- **[cloud-provider-vsphere]** ignore ept_rvi_mode and hv_mode
- **[docs]** Fix mentions of ingress-nginx module in docs.
- **[ingress-nginx]** Fix mentions of ingress-nginx module in docs.
- **[prometheus]** Fix mentions of ingress-nginx module in docs.

1.71.3

Features

- **[candi]** Add rosa 12.6.
- **[deckhouse-controller]** add validation for module source changes

Fixes

- **[candi]** containerd migration fix
- **[cni-cilium]** fixed hostPort workability with extraLoadBalancerAlgorithmsEnabled.
- **[log-shipper]** Fix matchExpressions values validation for ClusterLoggingConfig
- **[node-manager]** Hide Nvidia GPU dashboards if there is no GPU nodes in a cluster.
- **[node-manager]** containerd migration fix

1.71.2

Know before update

- Users will know how to configure KeyCloak and dexProvider to get user rights in dashboard and console.
- default
- if there are several ModuleSources with the prompp module available, the “deckhouse” ModuleSource will be used.

Features

- **[admission-policy-engine]** Added the ability to specify allowedServiceTypes and blockWildcardDomains in the SecurityPolicy object
- **[deckhouse-controller]** add leader election for non ha clusters
- **[user-authn]** Add allowedGroup in dexProvider OIDC

Fixes

- **[admission-policy-engine]** Fix validation policy for pods with skip-pss-check labels
- **[ci]** fix e2e creds
- **[ci]** fix e2e creds

- **[ci]** Run changelog check only on tag pushes to enforce release documentation
- **[ci]** create resources before deckhouse manifest and mutating webhook for prom rules for e2e tests
- **[ci]** Run changelog check only on tag pushes to enforce release documentation
- **[ci]** create resources before deckhouse manifest and mutating webhook for prom rules for e2e tests
- **[cloud-provider-aws]** Fix root_block_device converge.
- **[cni-cilium]** Fixed least-conn logs when feature is disabled
- **[deckhouse-controller]** fix new snapshot logic bag
- **[deckhouse-controller]** add more verbosity to snapshot error
- **[dhctl]** Fix false-positive staticinstance ip duplication preflight checks fails.
- **[docs]** Correction of KeyCloak documentation in d8-user-authn
  Users will know how to configure KeyCloak and dexProvider to get user rights in dashboard and console.
- **[docs]** Refactored table of supported OS versions.
- **[istio]** Added missed documentation about version 1.25
- **[metallb]** Fixed overwriting of Service `status` field by module components.
- **[metallb]** avoid double prefixes in the dashboard names.
  default
- **[node-manager]** Changed draining logic in update_approval
- **[node-manager]** fixed render nvidia-mig-manager
- **[prometheus]** use the “deckhouse” ModuleSource as the default for the prompp ModuleConfig if there are several ModuleSources with the prompp module available, the “deckhouse” ModuleSource will be used.
- **[runtime-audit-engine]** Improve memory footprint by switching to the stdlib memory allocator instead of jemalloc

1.71.1

Features

- **[docs]** Added documentation with recommendations for generating self-signed certificates

Fixes

- **[ci]** add sh symlink to bash to the deckhouse-controller
- **[ci]** Fix listing of changed modules.
- **[control-plane-manager]** correct indentation of the ‘certSANs’ in v1beta4 template.
- **[monitoring-ping]** Reducing PROCESSOR time consumption by changing the method of waiting for incoming packets.
- **[user-authn]** fix dex oidc connector insecureSkipVerify and rootCAData options

Version 1.70

Important

- The `ceph-csi` module has been removed. Use the `csi-ceph` module instead. Deckhouse will not be updated as long as `ceph-csi` is enabled in the cluster. For `csi-ceph` migration instructions, refer to the module documentation.
- Version 1.12 of the Ingress NGINX Controller has been added. The default controller version has been changed to 1.10. All Ingress controllers that do not have an explicitly specified version (via the `controllerVersion` parameter in the IngressNginxController resource or the `defaultControllerVersion` parameter in the `ingress-nginx` module) will be restarted.
- The `falco events` metric (from the `runtime-audit-engine` module) has been removed. The `falco events` metric was considered deprecated since DKP 1.68. Use the `falcosecurity falcosidekick_falco_events_total` metric instead. Dashboards and alerts based on the `falco_events` metric may stop working.
- All DKP components will be restarted during the update.

Major changes

- In the `Auto` update mode, patch version updates (for example, from `v1.70.1` to `v1.70.2`) are now applied taking into account the update windows, if they are set. Previously, in this update mode, only minor version updates (for example, from `v1.69.x` to `v1.70.x`) were applied with consideration to update windows, while patch version updates were applied as they appeared on a release channel.
- A node can now be rebooted if the corresponding Node object has the `update.node.deckhouse.io/reboot` annotation set.
- When cleaning up a static node, any local users created by Deckhouse Kubernetes Platform are

now also removed.
- Added synchronization monitoring for Istio in multi-cluster configurations. A new alert `D8IstioRemoteClusterNotSynced` has been introduced and triggers in the following cases:
  - The remote cluster is offline.
  - The remote API endpoint is not reachable.
  - The remote `ServiceAccount` token is invalid or expired.
  - There is a TLS or certificate issue between the clusters.
- The `deckhouse-controller collect-debug-info` command now also collects debug information for `Istio`, including:
  - Resources in the `d8-istio` namespace.
  - CRDs from the `istio.io` and `gateway.networking.k8s.io` groups.
  - `Istio` logs.
  - `Sidecar` logs of a single randomly selected user application.
- A new monitoring dashboard has been added to display OpenVPN certificate status. Upon expiration, server certificates will now be reissued, and client certificates will be removed. The following alerts have been added:
  - `OpenVPNClientCertificateExpired`: Warns about expired client certificates.
  - `OpenVPNServerCACertificateExpired`: Warns about an expired OpenVPN CA certificate.
  - `OpenVPNServerCACertificateExpiringSoon` and `OpenVPNServerCACertificateExpiringInAWeek`: Warn when an OpenVPN CA certificate is expiring in less than 30 or 7 days, respectively.
  - `OpenVPNServerCertificateExpired`: Warns about an expired OpenVPN server certificate.
  - `OpenVPNServerCertificateExpiringSoon` and `OpenVPNServerCertificateExpiringInAWeek`: Warn when an OpenVPN server certificate is expiring in less than 30 or 7 days, respectively.
- Monitoring dashboards have been renamed and updated:
  - “L2LoadBalancer” renamed to “MetalLB L2”; pool and column filtering added.
  - “Metallb” renamed to “MetalLB BGP”; pool and column filtering added. The ARP request panel has been removed.
  - “L2LoadBalancer / Pools” renamed to “MetalLB / Pools”.
- The `upmeter` module’s PVC size has been increased to accommodate data retention for 13 months. In some cases, the previous PVC size was insufficient.
- The ModuleSource resource status now includes information about module versions in the source.
- The Module resource status now includes information about the module’s lifecycle stage. A module can move through the following stages in its lifecycle: Experimental, Preview, General Availability, and Deprecated. For details on module lifecycle stages and how to evaluate its stability, refer to the corresponding section in the documentation.
- It is now possible to use stronger or more modern encryption algorithms (such as `RSA-3072`, `RSA-4096`, or `ECDSA-P256`) for control plane cluster certificates instead of the default `RSA-2048`. You can use the `encryptionAlgorithm` parameter in the ClusterConfiguration resource to configure this.
- The `descheduler` module can now be configured to evict pods that are using local storage. Use the `evictLocalStoragePods` parameter in the module configuration to adjust this.
- You can now adjust the logging level of the Ingress controller using the `controllerLogLevel` parameter in the IngressNginxController resource. The default log level is `Info`. Controlling the logging level can help prevent log collector overload during Ingress controller restarts.

Security

- The severity level of alerts indicating security policy violations has been raised from 7 to 3.
- The configuration for `Yandex Cloud`, `Zvirt`, and `Basis Dynamix` providers now uses `OpenTofu` instead of `Terraform`. This enables easier provider updates, such as applying fixes for known vulnerabilities (CVEs).
- CVE vulnerabilities have been fixed in the following modules: `chrony`, `descheduler`, `dhctl`, `node-manager`, `registry-packages-proxy`, `falco`, `cni-cilium`, and `vertical-pod-autoscaler`.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.70.19

Fixes

- **[ci]** Fix render workflow job.
- **[istio]** Reduce CPU and RAM for regenerate multicluster JWT token and sort ingressGateway

Chore

- **[runtime-audit-engine]** Update CI file

1.70.18

Fixes

- **[istio]** The metrics-exporter's template is fixed, it blocked the main queue if `controlPlane.nodeSelector` setting was configured.

1.70.17

Fixes

- **[istio]** fixed an error in processing custom input gateway addresses using the configuration option of the alliance.ingressGateway.advertise module.

1.70.16

Fixes

- **[cloud-provider-aws]** fix root_block_device converge
- **[runtime-audit-engine]** Improve memory footprint by switching to the stdlib memory allocator instead of jemalloc

1.70.15

Fixes

- **[control-plane-manager]** correct indentation of the 'certSANs' in v1beta4 template.

1.70.14

Fixes

- **[cloud-provider-dvp]** fix logic of work with disks and coreFraction validation
- **[cloud-provider-huaweicloud]** fix virtualMachineClassName in terraform
- **[deckhouse]** Filter deleted pvcs in storage class change hook.
- **[deckhouse]** Filter deleted pvcs in storage class change hook.
- **[deckhouse-controller]** make metrics more verbose

1.70.13

Know before update

- Users can now restore etcd backups and scale their master nodes efficiently without encountering previous errors.

Fixes

- **[control-plane-manager]** Fixed issues with etcd backup restoration process, ensuring reliable master node scaling from 1 to 3.
  Users can now restore etcd backups and scale their master nodes efficiently without encountering previous errors.
- **[deckhouse-controller]** fix nil annotations map for module fetcher
- **[ingress-nginx]** Validation is switched off temporarily.
- **[prometheus]** Fix fake longterm emptydir alert.

1.70.12

Features

- **[cloud-provider-huaweicloud]** add enterprise_project_id

Fixes

- **[dhctl]** TLS support on dhctl edition checking step.

1.70.11

Fixes

- **[docs]** remove Yandex Cloud’s api-gateway.editor and connection-manager.editor roles

1.70.10

Features

- **[docs]** Add unified manual about edition switch

Fixes

- **[dhctl]** Fix dhctl output in case of invalid config
- **[dhctl]** Release infrastructure dependency lock.

1.70.9

Fixes

- **[control-plane-manager]** Fix default behaviour for etcd backup.
- **[ingress-nginx]** fixed geoip issues in validator
- **[ingress-nginx]** Automatic geoip updating feature was reverted.

1.70.8

Fixes

- **[docs]** Delete the mention that addresses are allocated starting from 10 from internalNetworkCIDR in vcd cloud-provider

1.70.7

Know before update

- When called collect-debug-info command in cluster without Istio, user see errors, but archive create correct

Fixes

- **[control-plane-manager]** fix kubeadm v1beta4 config
- **[deckhouse-controller]** fix wrong behavior of Pending ModuleReleases
- **[ingress-nginx]** fixed `node-selector` and `tolerations` in validator `pod`
- **[istio]** Fix for collecting debugging resources in collect-debug-info about Istio
  When called collect-debug-info command in cluster without Istio, user see errors, but archive create correct
- **[node-local-dns]** Improved stability for clusters with poor node-to-node connection quality by increasing node-local-dns to kube-dns healthcheck timeouts.

1.70.6

Fixes

- **[control-plane-manager]** audit-log-maxage value to 30 days
- **[deckhouse]** Fix obsolete config alert stuck after deleting module config.
- **[deckhouse]** Remove unnecessary modules downloading.
- **[docs]** Rendered SSHHost and SSHConfig specs.
- **[ingress-nginx]** fixed an issue with the snapshot retrieval mechanism
- **[ingress-nginx]** fix missing proxy client cert in validator deployment
- **[ingress-nginx]** Ingress resource validation in ingress-nginx controller re-enabled.
- **[openvpn]** Added a migration hook to mitigate false positive `OpenVPNClientCertificateExpired` alerts.
- **[user-authz]** Add patch verb for service accounts

1.70.5

Know before update

- loki statefulset will be rollout restarted

Fixes

- **[loki]** Fix several bugs introduced while implementing loki disk based retention
  loki statefulset will be rollout restarted
- **[prometheus]** Add headers field to values spec

1.70.4

Know before update

- operator-prometheus

Features

- **[cloud-provider-dvp]** Add vmClassName to the config
- **[operator-prometheus]** Add to documetation for operator-prometheus CRD. operator-prometheus
- **[snapshot-controller]** Automatic snapshot-controller MC creation for future switch to external module

Fixes

- **[cloud-provider-dvp]** Fix cherry-pick conflicts pr
- **[ingress-nginx]** Ingress resource validation in ingress-nginx controller re-enabled.

1.70.3

Fixes

- **[cloud-provider-yandex]** Handle hybrid clusters case in the to_tofu_migrate_metric hook
- **[cloud-provider-zvirt]** fix UUID validation regex
- **[deckhouse-controller]** fixes default maintenance value which prevents stuck metric
- **[deckhouse-tools]** deckhouse-tools fix deployment render
- **[docs]** Updated conversions message.
- **[node-local-dns]** Changed container liveness port.
- **[node-manager]** fixed update for static clusters

1.70.2

Fixes

- **[candi]** memory.available calculation python2 fix
- **[openvpn]** Fixed condition of alert triggering for client certificate.
- **[service-with-healthchecks]** change port kube-rbac-proxy in agent DaemonSet

1.70.1

Know before update

- prometheus

Features

- **[cloud-provider-dvp]** Add vmClassName to the config
- **[prometheus]** Validation prometheus remotewrite ca webhook prometheus

Fixes

- **[candi]** Added a check to the bashible step for the presence of a previously created symlink and where it leads to
- **[candi]** bashible bb-discover-node-name use node name from kubelet certificate
- **[ci]** returned the condition when the branch was not found
- **[ci]** fix kubernetes version info in the release file
- **[cloud-provider-vcd]** Fix unexpected behavior with legacy API versions in VCD
- **[deckhouse-controller]** fix deckhouse crds mechanism
- **[dhctl]** Fix unexpected behavior with legacy API versions in VCD
- **[docs]** Update docs about Keycloak configuration for IP
- **[operator-trivy]** Add support anon registry creds for bdu updater.
- **[user-authz]** fix error with webhook system-roles

Version 1.69

Important

- Support for Kubernetes 1.32 has been added, while support for Kubernetes 1.27 has been discontinued. The default Kubernetes version has been changed to 1.30. In future DKP releases, support for Kubernetes 1.28 will be removed.

- All DKP components will be restarted during the update.

Major changes

- The `ceph-csi` module is now deprecated. Plan to migrate to the `csi-ceph` module instead. For details, refer to the Ceph documentation.
- You can now grant access to Deckhouse web interfaces using user names via the `auth.allowedUserEmails` field. Access restriction is configured together with the `auth.allowedUserGroups` parameter in configuration of the following modules with web interfaces: `cilium-hubble`, `dashboard`, `deckhouse-tools`, `documentation`, `istio`, `openvpn`, `prometheus`, and `upmeter` (example for `prometheus`).
- A new dashboard **Cilium Nodes Connectivity Status & Latency** has been added to Grafana in the `cni-cilium` module. It helps monitor node network connectivity issues. The dashboard displays a connectivity matrix similar to the `cilium-health status` command, using metrics that are already available in Prometheus.
- A new `D8KubernetesStaleTokensDetected` alert has been added in the `control-plane-manager` module that is triggered when stale service account tokens are detected in the cluster.
- You can now create a Project from an existing namespace and adopt existing objects into it. To do this, annotate the namespace and its resources with `projects.deckhouse.io/adopt`. This lets you switch to using Projects without recreating cluster resources.
- A `Terminating` status has been added to ModuleSource and ModuleRelease resources. The new status will be displayed when an attempt to delete one of them fails.
- The installer container now automatically configures cluster access after a successful bootstrap. A `kubeconfig` file is generated in `~/.kube/config`, and a local TCP proxy is set up through an SSH tunnel. This allows you to use kubectl locally right away without manually connecting to the control-plane node via SSH.
- Changes to Kubernetes resources in multi-cluster and federation setups are now tracked directly via Kubernetes API. This enables faster synchronization between clusters and eliminates the use of outdated certificates. In addition, mounting of ConfigMap and Secret resources into Pods has been removed to eliminate family system compromise risks.
- A new dynamicforward plugin has been added to CoreDNS, improving DNS query processing in the cluster. It integrates with `node-local-dns`, continuously monitors `kube-dns` endpoints, and automatically updates the list of DNS forwarders. If the control-plane node is unavailable, DNS queries are still forwarded to available endpoints, improving cluster stability.
- A new log rotation approach has been introduced in the `loki` module. Now, old logs are automatically removed when disk usage exceeds a threshold: either 95% of PVC size or PVC size minus the size required to store two minutes of log data at the configured ingestion rate (`ingestionRateMB`). The `retentionPeriodHours` parameter no longer controls the data retention and is used for monitoring alerts only. If `loki` begins removing old logs before the set period is reached, a `LokiRetentionPerionViolation` alert will be triggered, informing the user that they must reduce the value of `retentionPeriodHours` or increase the PVC size.
- A new `nodeDrainTimeoutSecond` parameter lets you set the maximum timeout when attempting to drain a node (in seconds) for each NodeGroup resource. Previously, you could only use the default value (10 minutes) or reduce it to 5 minutes using the `quickShutdown` parameter, which is now deprecated.
- The `openvpn` module now includes a `defaultClientCertExpirationDays` parameter, allowing you to define the lifetime of client certificates.

Security

- Known vulnerabilities have been addressed in the following modules: `ingress-nginx`, `istio`, `prometheus`, and `local-path-provisioner`.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.69.17

Know before update

- Users can now configure containerd to pull images from private registries, easing deployments in private environments.

Features

- **[docs]** Add command to view all falco audit rules to docs.
- **[istio]** Validating jwt token exp time less than 30 days
- **[node-manager]** Added support for configuring private HTTP container registries in containerd. Users can now configure containerd to pull images from private registries, easing deployments in private environments.

Fixes

- **[docs]** Removed port 22322 from setup docs.
- **[istio]** Fix istiod CPU leak when the order of the ingressGateways array is changed
- **[istio]** Reduce RAM for regenerate multicluster JWT token

1.69.16

Fixes

- **[control-plane-manager]** audit-log-maxage value to 30 days
- **[deckhouse]** Fix obsolete config alert stuck after deleting module config.

1.69.15

Know before update

- Prevent important data loss when using wide retention intervals.

Fixes

- **[loki]** Fix incorrect indices sort function used in disk-based retention.
  Prevent important data loss when using wide retention intervals.

1.69.14

Fixes

- **[deckhouse-controller]** fixes default maintenance value which prevents stuck metric
- **[node-manager]** fixed update for static clusters

1.69.13

Fixes

- **[istio]** The `alliance.ingressGateway.advertise` option was fixed up.

1.69.12

Fixes

- **[dashboard]** fix display workload metrics on dashboard

1.69.11

Know before update

- In L2 mode, the Public IP address will not be marked as free in the pool after deleting the LoadBalancer Service. As a temporary fix, you can restart the MetalLB controller pods.
- fix static e2e tests for fe/upgrade

Fixes

- **[ci]** fix static e2e tests for fe/upgrade
  fix static e2e tests for fe/upgrade
- **[istio]** proxy-buffer-size increased in kiali Ingress.
- **[metallb]** Reverted the fix for handling deleted Services and revising the IP pools in L2 mode. The correct fix is under construction.
  In L2 mode, the Public IP address will not be marked as free in the pool after deleting the LoadBalancer Service. As a temporary fix, you can restart the MetalLB controller pods.

1.69.10

Fixes

- **[deckhouse-controller]** Added validation according to `RFC 1123` for module names added to `ModuleSource`.
- **[docs]** Fix apiVersion for s3 storageclass

1.69.9

Fixes

- **[metallb]** Fixed deprecated CRD addresspools.metallb.io deletion.
- **[network-gateway]** Fix python path

1.69.8

Fixes

- **[cloud-provider-yandex]** fix synchronizing TGs
- **[deckhouse]** Gracefully restore deployed modules on HA installations.

1.69.7

Fixes

- **[docs]** Fixed new layouts at documentation.
- **[kube-dns]** Fixed release upgrade issue with removed d8-kube-dns-redirect service.

1.69.6

Fixes

- **[cni-cilium]** Fixed race condition when deleting element from ipcache map during VM migration.
- **[docs]** Fix broken links in the Storage overview.
- **[metallb]** Fixed IP pool exhaustion on LoadBalancer deletion.
- **[node-manager]** Revert cluster autoscaler

1.69.5

Features

- **[candi]** Add rosa 12.6.

Fixes

- **[cloud-provider-yandex]** fix WithNATInstance clusters converge
- **[control-plane-manager]** Stale service account alert upd
- **[node-manager]** Autoscaler - remove additional cordon node by mcm provider.

1.69.4

Features

- **[deckhouse-controller]** sequential processing of module releases
- **[deckhouse-controller]** sequential processing of module releases
- **[testing]** Add e2e tests for autoscaler and enable it permanent for GCP.
- **[testing]** Add e2e tests for autoscaler and enable it permanent for GCP

Fixes

- **[candi]** Return support `.cri.notManaged.criSocketPath`.
- **[deckhouse]** Restore only ready mpo.
- **[dhctl]** <Disable caching metaconfig during converge and disable converge deckhouse manifests.
- **[dhctl]** Fix incorrect behavior that fails sudoers preflight check if password contains whitespaces.
- **[docs]** Fixed Dynamix layout at documentation.
- **[docs]** Add support of `x-doc-d8Editions` array instead of `x-doc-d8Revision` field to JSON schema renderer.
- **[node-manager]** Increase verbosity and revert versions and fix bug with unsupported mcm annotation
- **[upmeter]** upmeter sa token rotation
- **[user-authn]** Add separate ServiceAccount to basic-auth-proxy.
- **[user-authz]** Exclude deckhouse clusterrolebindings from webhook.

1.69.3

Know before update

- Ingress-nginx controller pods of v1.9 will be restated.

Fixes

- **[control-plane-manager]** Stale service account alert fix.
- **[deckhouse]** Limit module name.
- **[deckhouse]** Fix mpo/source requeue.
- **[docs]** Add info about debug GrafanaDashboardDefenition.
- **[extended-monitoring]** Fix x509_certificate_exporter_resources deployment
- **[ingress-nginx]** All necessary shared libraries are added to the container image.
  Ingress-nginx controller pods of v1.9 will be restated.
- **[kube-dns]** Expanded pattern for `stubZones` to allow underscores (`_`) in domain names.
- **[user-authn]** Fix secret generation on empty data field in the dex client app secret.

1.69.2

Know before update

- the grafana deployment will be rollout restarted

Fixes

- **[deckhouse]** Gracefully restore deployed modules on HA installations.
- **[prometheus]** enable WAL for the grafana SQLite database to prevent locking errors, thus fixing in-dashboard alerting.
  the grafana deployment will be rollout restarted

1.69.1

Know before update

- fixes the bug introduced in v1.69.0
- release upgrade will be blocked on AWS-based clusters where SA doesn't have DescribeAddressesAttribute and DescribeInstanceTopology roles. They are required for new Terraform AWS Provider version.

Features

- **[deckhouse]** enable UnmetCloudConditions check
  release upgrade will be blocked on AWS-based clusters where SA doesn't have DescribeAddressesAttribute and DescribeInstanceTopology roles. They are required for new Terraform AWS Provider version.
- **[deckhouse-controller]** rename module flag `managementState` to `maintenance`.
- **[dhctl]** add variable time range to request counter
- **[node-manager]** enable UnmetCloudConditions check
- **[node-manager]** UnmetCloudConditions requirement and alert

Fixes

- **[ci]** Use fixed werf version.
- **[cloud-provider-vcd]** Add a hook to set `legacyMode` based on the detected `VCD API` version
- **[deckhouse]** Clear deployed release`s message.
- **[deckhouse]** Delete deckhouse epslices.
- **[deckhouse-controller]** Fix runtime handling for the global config.
- **[dhctl]** fix bootstrap and abort config preparation
- **[docs]** Fix d8 install commands.
- **[loki]** fix storage capacity calculator hook for Loki
  fixes the bug introduced in v1.69.0
- **[operator-trivy]** Add proxy env variables support to the trivy server.

Version 1.68

Important

- After the update, the UID will change for all Grafana data sources created using the GrafanaAdditionalDatasource resource. If a data source was referenced by UID, that reference will no longer be valid.

Major changes

- A new parameter, `iamNodeRole`, has been introduced for the AWS provider. It lets you specify the

name of the IAM role to bind to all AWS instances of cluster nodes. This can come in handy if you need to grant additional permissions (for example, access to ECR, etc.).

- Creating nodes of the CloudPermanent type now takes less time. Now, CloudPermanent nodes are created in parallel. Previously, they were created in parallel only within a single group.
- Monitoring changes:
  - Support for monitoring certificates in secrets of the `Opaque` type has been added.
  - Support for monitoring images in Amazon ECR has been added.
  - A bug that could cause partial loss of metrics when Prometheus instances were restarted has been fixed.
- When using a multi-cluster Istio configuration or federation, you can now explicitly specify the list of addresses used for inter-cluster requests. Previously, these addresses were determined automatically; however, in some configurations, they could not be resolved.
- The DexAuthenticator resource now has a `highAvailability` parameter that controls high availability mode. In high availability mode, multiple replicas of the authenticator are launched. Previously, high availability mode of all authenticators was determined by a global parameter or by the `user-authn` module. All authenticators deployed by DKP now inherit the high availability mode of the corresponding module.
- Node labels can now be added, removed, or modified using files stored on the node in the `/var/lib/node_labels` directory and its subdirectories. The full set of applied labels is stored in the `node.deckhouse.io/last-applied-local-labels` annotation.
- Support for the Huawei Cloud provider has been added.
- The new `keepDeletedFilesOpenedFor` parameter in the `log-shipper` module allows you to configure the period to keep the deleted log files open. This way, you can continue reading logs from deleted pods for some time if log storage is temporarily unavailable.
- TLS encryption for log collectors (Elasticsearch, Vector, Loki, Splunk, Logstash, Socket, Kafka) can now be configured using secrets, rather than by storing certificates in the ClusterLogDestination resources. The secret must reside in the `d8-log-shipper` namespace and have the `log-shipper.deckhouse.io/watch-secret: true` label.
- In the project status under the `resources` section, you can now see which project resources have been installed. Those resources are marked with `installed: true`.
- A new parameter, `--tf-resource-management-timeout`, has been added to the installer. It controls the resource creation timeout in cloud environments. By default, the timeout is set to 10 minutes. This parameter applies only to the following clouds: AWS, Azure, GCP, OpenStack.

Security

Known vulnerabilities have been addressed in the following modules:

- `admission-policy-engine`
- `chrony`
- `cloud-provider-azure`
- `cloud-provider-gcp`
- `cloud-provider-openstack`
- `cloud-provider-yandex`
- `cloud-provider-zvirt`
- `cni-cilium`
- `control-plane-manager`
- `extended-monitoring`
- `descheduler`
- `documentation`
- `ingress-nginx`
- `istio`
- `loki`
- `metallb`
- `monitoring-kubernetes`
- `monitoring-ping`
- `node-manager`
- `operator-trivy`
- `pod-reloader`
- `prometheus`
- `prometheus-metrics-adapter`
- `registrypackages`
- `runtime-audit-engine`
- `terraform-manager`

- `user-authn`
- `vertical-pod-autoscaler`
- `static-routing-manager`

Mandatory component restart

The following components will be restarted after updating DKP to 1.68:

- Kubernetes Control Plane
- Ingress controller
- Prometheus, Grafana
- `admission-policy-engine`
- `chrony`
- `cloud-provider-azure`
- `cloud-provider-gcp`
- `cloud-provider-openstack`
- `cloud-provider-yandex`
- `cloud-provider-zvirt`
- `cni-cilium`
- `control-plane-manager`
- `descheduler`
- `documentation`
- `extended-monitoring`
- `ingress-nginx`
- `istio`
- `kube-state-metrics`
- `log-shipper`
- `loki`
- `metallb`
- `monitoring-kubernetes`
- `monitoring-ping`
- `node-manager`
- `openvpn`
- `operator-trivy`
- `prometheus`
- `prometheus-metrics-adapter`
- `pod-reloader`
- `registrypackages`
- `runtime-audit-engine`
- `service-with-healthchecks`
- `static-routing-manager`
- `terraform-manager`
- `user-authn`
- `vertical-pod-autoscaler`

The complete list of changes, including the updated components, is available in the changelog on GitHub.

1.68.15

Fixes

- **[dhctl]** Do not create a local-converger lock for commanderMode
- **[docs]** Fixed the parameter generator for modules on Hugo.
- **[docs]** This reverts commit cf9d7590da7c4b23c7eab10da37abc16397faf87.

1.68.14

Fixes

- **[deckhouse-controller]** fix module release force behaviour
- **[deckhouse-controller]** fix module release force behaviour
- **[runtime-audit-engine]** Fix CrashLoopBack due to invalid config.

1.68.13

Know before update

- Grafana deployment will be rollout restarted.

Fixes

- **[prometheus]** Enables WAL for Grafana SQLite database to prevent locking errors, thus fixing in-

dashboard alerting.
Grafana deployment will be rollout restarted.

1.68.12

Fixes

- **[ci]** Use fixed werf version.
- **[cloud-provider-zvirt]** fix invalid zvirt csi template
- **[node-local-dns]** Stale-dns-connection-cleaner file permissions and security context are fixed.

1.68.11

Fixes

- **[cloud-provider-zvirt]** fix zvirt-csi-driver patching (token refresh fix patch)
- **[node-manager]** partial rollback of changes in images cluster-autoscaler

1.68.10

Features

- **[deckhouse-controller]** restore releases only by deckhouse-controller and in deployed state

Fixes

- **[deckhouse-controller]** fix suspend release
- **[node-manager]** handle_node_template hook ignore metallb label
- **[runtime-audit-engine]** copy falcosecurity_falcosidekick_falco_events_total metric to falco_events for backward compatibility

1.68.9

Fixes

- **[node-manager]** partial rollback of changes in images cluster-autoscaler

1.68.8

Features

- **[cert-manager]** Bump cert-manager version up to 1.17.1. This fixes acme confirmation for cloudflare provider.

Fixes

- **[docs]** FIx local docimentation build and run.
- **[docs]** The information label with the editions has been removed from the global settings page.
- **[ingress-nginx]** Fixed the display of ports in the examples in the CRDs.
- **[istio]** Fixed the display of ports in the examples in the CRDs.

1.68.7

Know before update

- Ingress-nginx pods will restart. The validation of the generated NGINX configuration during the validation of `Ingress` resources is disabled until finding a way of running validations safely.

Fixes

- **[ingress-nginx]** latest CVE fixes backported.
  Ingress-nginx pods will restart. The validation of the generated NGINX configuration during the validation of `Ingress` resources is disabled until finding a way of running validations safely.

1.68.6

Fixes

- **[candi]** Fix errors in bashible steps.
- **[deckhouse]** Fix erroneous inclusions in 1.68
- **[deckhouse]** Fix module config ignored.
- **[deckhouse]** Fix checking automatic kubernetes version in managed clusters.
- **[deckhouse]** Fix module config ignored.

- **[deckhouse]** Explicitly set ports in d8-kube-dns deployment.
- **[deckhouse]** Set EnabledByModuleConfig condition by default for modules without this condition.
- **[deckhouse]** Fix module condition race.
- **[deckhouse-controller]** Fix applying releases and their metrics.
- **[deckhouse-controller]** Fix applying releases and their metrics.
- **[deckhouse-tools]** d8-cli build fix
- **[docs]** Corrected a typo in the NFSStorageClass name in the NFS data storage section.
- **[docs]** Added SE and SE+ restrictions for Istio.
- **[log-shipper]** Add delete parsed_data to final kafka transform render

1.68.5

Features

- **[prometheus]** add grafana enable flag

Fixes

- **[candi]** Fix `disable_unattended_upgrades` bashible step
- **[docs]** Correct mistakes in the crds for the Storage section in Deckhouse Kubernetes Platform.
- **[docs]** Update Deckhouse CLI installation guide
- **[docs]** Fixed sidebar behaviour with links.
- **[docs]** Correct mistakes in the documentation for the storage section in Deckhouse Kubernetes Platform.
- **[extended-monitoring]** Minor certificate alerts improvements
- **[metallb]** Add clearing of alert metric group D8MetallbObsoleteLayer2PoolsAreUsed
- **[operator-trivy]** Updated info about databases with vulnerabilities.

1.68.4

Fixes

- **[candi]** Fix add_node_users.sh search for sudoersd_path.
- **[deckhouse-controller]** Fix DeckhouseRelease approval in the Manual update mode.
- **[dhctl]** Escape special shell characters in sudo preflight check.
- **[docs]** Fixed typos at VMware configuration parameters.
- **[upmeter]** Add D8UpmeterDiskUsage alert.

1.68.3

Fixes

- **[deckhouse]** Add initial module config validation.
- **[deckhouse-controller]** Update addon-operator.
- **[dhctl]** sudoPassword from the connection-config not used in dhctl CLI and dhctl server
- **[metallb]** Fixed module startup error with empty Selector value in SVC.

1.68.2

Know before update

- The links in the alerts will be correct when the documentation module is enabled.

Fixes

- **[admission-policy-engine]** Fix wildcard apparmor profile.
- **[deckhouse-controller]** fix deckhouse-release check loop
- **[deckhouse-tools]** Fix tools links pointing to wrong path
- **[metallb]** Fixed alerts for EE edition.
- **[monitoring-applications]** Fix module dependencies. The `monitoring-applications` module was affected and disabled by the enabled script.
- **[node-local-dns]** Fix module dependencies. The `monitoring-applications` module was affected and disabled by the enabled script.
- **[prometheus]** Changes the links in alert’s template about alerts from url to relative links
  The links in the alerts will be correct when the documentation module is enabled.
- **[service-with-healthchecks]** Fixed several issues.
- **[upmeter]** Fixed metallb module probe.

1.68.1

Know before update

- kubernetes-api will be restarted.

- okmeter daemonset pods will be rollout restarted

Features

- **[candi]** Add a new layout for HuaweiCloud provider with support for VPC peering connections.
- **[docs]** Add new description info for conversions.

Fixes

- **[candi]** Fix update_node_labels.sh to work on python2.
- **[candi]** Reduce `request-timeout` in kubernetes-api to 60s.
  kubernetes-api will be restarted.
- **[ci]** CI_COMMIT_TAG does not found in deckhouse controller
- **[ci]** fix standalone-installer build
- **[ci]** cloud providers CVE fixes, use single images for cloud-provider-zvirt and cloud-provider-yandex
- **[cloud-provider-vcd]** fix CVE, bump ccm, use go 1.23, use werf images and src-artifact
- **[deckhouse]** Fix stuck overridden status in ms.
- **[deckhouse]** Fix module.yaml ignored when weight == 0.
- **[node-manager]** fix autoscaler RBAC for zvirt and dynamix
- **[okmeter]** drop service account fields from the okmeter daemonset manifest
  okmeter daemonset pods will be rollout restarted
- **[user-authn]** Fix cronjob security context.

Embedded Deckhouse Platform modules

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The admission-policy-engine module

> Available with limitations in: **Open/CE, Core, BE, SE, SE+**
>
> Available without limitations in: **Ultimate/EE**

> Included in extensions: **Advanced Infrastructure Security**

> The module lifecycle stage: General Availability

The `admission-policy-engine` module implements support for admission security policies in a Kubernetes cluster.

Admission policies are rules applied to objects (e.g., Pod and Service) at the time of their creation or modification in the cluster (but not during their operation), based on the information provided in their manifest. These policies are aimed at formalizing parameters that are allowed or prohibited in object manifests.

Policies are divided into three categories:

- Pod Security Standards: Policies that comply with the relevant Pod Security Standards.
- Operational policies: Policies for creating additional requirements for objects by validating the values of parameters that are **not directly related** to security (for example, a list of allowed prefixes for container images, an image download policy, a list of required container images, etc.).
- Security policies: Policies for creating additional requirements on objects by validating the values of security-related parameters (e.g., container access to the host's IPC or PID namespaces, privilege lists for containers, etc.).

> These policies complement each other. If multiple policies are applied to a single namespace, objects are validated against each of them. If even one policy is violated, the object will not be created.

In addition to policies that prohibit using parameters different from the set requirements, the module supports the SecurityPolicyException resource, which allows creating fine-grained exceptions from security policy checks. With this resource, you can allow using specific parameters for individual pods or containers without changing security policies applied to the entire namespace.

How validation failure messages are displayed

Depending on how pods are created, there are differences in how the API generates messages regarding validation failures (violations of established policies):

- If a pod is created directly, the validation error is returned in the API response indicating a validation failure (policy violation).
- If pods are created via a Deployment and the pods potentially violate a policy, the creation of such Deployment will be rejected.
- If pods are created via a Deployment that already existed before the policy was enabled, the required number of ReplicaSets is created, which in turn attempt to create the pods. In this case, the validation error is not returned in the API response but is displayed in the namespace events or the corresponding ReplicaSet events.

Controller-level validation

Policies validate pod-creating controllers at `CREATE` and `UPDATE` time. This provides early feedback when a workload is created or updated, before any Pod is launched, and also surfaces creation denials when deploying via CI (CI rarely creates Pods directly, and Pod creation errors from controllers are often not displayed).

⚠ By default, a controller that violates a policy with `enforcementAction: Deny` gets a warning instead of a denial. The Pods the controller creates are still denied. The module checks controllers in separate constraints named `d8-controllers-<NAME>` with `enforcementAction: warn`. The `ControllerPolicyViolation` alert reports such controllers, one alert for each policy and namespace. This is a temporary default for the transition to controller-level validation. To check controllers with the action of the policy now, set `controllerValidationEnforcement` to `Policy`. In a future release, `Policy` will become the default.

The following objects are validated:

API group	Resource kind	Operations
(core)	Pod	`CREATE`, `UPDATE`, `DELETE`
apps	Deployment, StatefulSet, DaemonSet	`CREATE`, `UPDATE`, `DELETE`
(core)	ReplicationController	`CREATE`, `UPDATE`, `DELETE`
batch	Job, CronJob	`CREATE`, `UPDATE`, `DELETE`
(core)	pods/exec, pods/attach	`CONNECT`

`DELETE` is intercepted so that a policy can forbid deleting a resource: a rule that inspects `input.review.operation` only runs if the request reaches Gatekeeper.

⚠ The webhook uses `failurePolicy: Fail`. While Gatekeeper is unavailable, both creating and deleting the resource kinds listed above is blocked. To restore operations, bring the `gatekeeper-controller-manager` deployment back up.

⚠ Controller-level checks run against the **original pod template** (`spec.template`), not against the manifest that results from all mutations. Pod-level validation, in contrast, happens after all mutating actions — for example, after LimitRange adds default `resources`, or a mutating webhook adds `securityContext`. Keep this in mind when using security policies.

labelSelector semantics for controllers

When building policies that are applied based on object labels (`match.labelSelector`), keep in mind that selection is performed against the `metadata.labels` of the reviewed object. For Pods these are the pod's labels. For controllers (Deployment, StatefulSet, and others) these are the controller's **top-level** `metadata.labels`, not the pod template's `metadata.labels` (`spec.template.metadata.labels`).

As a result, a selector that matches the pods a controller creates does not by itself trigger a controller-level check, and an exclusion selector (`NotIn`/`DoesNotExist`) that excludes those pods may not take effect at the controller level. In both cases the controller itself must be matched by the `labelSelector`.

This also applies to the `security.deckhouse.io/skip-pss-check` label, which excludes an object from the Pod Security Standards constraints. The label is read from the reviewed object's own `metadata.labels`, so a workload that carries it only on the pod template (`spec.template.metadata.labels`) is exempt at the Pod level but still checked at the controller level. To exempt such a workload completely, set the label on the controller's top-level `metadata.labels` as well. Unlike SecurityPolicyException, this cannot be resolved from the pod template: the exclusion is evaluated by Gatekeeper's `match.labelSelector`, before any policy code runs.

SecurityPolicyException label resolution for controllers

For controller kinds, SecurityPolicyException (SPE) labels and annotations are taken from the **pod template's metadata** (`spec.template.metadata`), not from the controller's top-level metadata. This ensures SPEs apply to the pods the controller creates, not to the controller object itself.

Configuring controller-level validation

Controller-level validation is configured by four settings with different scopes:

1. The `controllerValidation` module parameter enables or disables controller-level validation for every constraint that reads a pod spec. Despite living in the `podSecurityStandards` section, it applies to Pod Security Standards, OperationPolicy and SecurityPolicy alike.
2. The `controllerValidationEnforcement` module parameter sets what happens to a controller that violates a denying policy: `Warn` (default) accepts it with a warning, `Policy` denies it. It applies to the Pod Security Standards and to every OperationPolicy and SecurityPolicy without `spec.controllerValidation.mode`.
3. The `controllerValidationExcludedPolicies` module parameter lists the Pod Security Standards rules that are checked on Pods only. It applies to the Pod Security Standards only.
4. The `spec.controllerValidation` section of an OperationPolicy or a SecurityPolicy configures that policy.

`mode: Enforce` checks controllers with the action of the policy, `mode: Warn` only warns about them, and `mode: Disabled` checks only Pods. `Enforce` and `Warn` check controllers even when the module parameter disables it, and a policy without `mode` follows the module parameters. `excludedPolicies` lists the rules of the policy that are checked on Pods only; the module exclusion list does not apply to a policy.

Exclude a rule from controller-level validation when a mutating webhook makes the Pod compliant while the controller's pod template is not. For example, the Vertical Pod Autoscaler sets `resources.requests` on the Pod, while the pod template may set `limits` only. A container that sets no `resources` at all already passes the controller-level check in the lenient mode, so the exclusion matters when `resources` are set partially. The following OperationPolicy checks the image registry on controllers and the resource requests on Pods only:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: requests-set-by-vpa
spec:
  enforcementAction: Deny
  controllerValidation:
    mode: Enforce
    excludedPolicies:
      - requiredResources
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/enabled: "true"
  policies:
    allowedRepos:
      - registry.example.com
    requiredResources:
      requests:
        - cpu
        - memory
```

A Deployment whose pod template sets `limits` but no `requests` is accepted, and its Pods are denied unless the Vertical Pod Autoscaler sets the requests.

The rules of the Pod Security Standards are excluded in the module settings. They are named as the parameters of the SecurityPolicy objects the standards are published with, see Rules of the standards. The following example checks `allowPrivilegeEscalation` and `requiredDropCapabilities` on Pods only:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  version: 1
  settings:
    podSecurityStandards:
      controllerValidationExcludedPolicies:
        - allowPrivilegeEscalation
        - requiredDropCapabilities
```

Some rules are checked by one constraint and are excluded together: listing any rule of a group checks the whole group on Pods only. The groups are:

- `allowHostPID` and `allowHostIPC`;
- `allowHostNetwork` and `allowedHostPorts`;
- `fsGroup`, `runAsUser`, `runAsGroup` and `supplementalGroups`;
- `allowedUnsafeSysctls` and `forbiddenSysctls`.

`allowedCapabilities` and `requiredDropCapabilities` are excluded one by one.

When a rule is checked on Pods only:

- controllers (Deployment, StatefulSet, DaemonSet, Job, CronJob, ReplicationController) are not checked against it on creation or update;
- SecurityPolicyException labels on `spec.template.metadata.labels` of controllers are not resolved for it, since it is not applied to controllers;
- Pods are still checked against it at launch time.

Lenient mode for fields that may be filled in before the pod is created

Some constraints use a **lenient mode** for fields that may be absent from a controller's pod template and still be present on the Pod the controller creates. This prevents denying a workload whose pods would in fact be compliant.

The following constraints skip violations for absent fields when reviewing controllers, while still enforcing them when a field is **explicitly set** to a disallowed value. The decision is made per container: a container that omits the field is treated leniently even if another container in the same pod template sets it.

Constraint	Skipped fields (when absent on controllers)	What may fill the field in
`D8RequiredResources`	`container.resources` (when neither `limits` nor `requests` is set)	LimitRange
`D8AllowedUsers`	`runAsUser`, `runAsGroup`, `fsGroup`, `supplementalGroups` (with `MustRunAs` / `MustRunAsNonRoot`)	A mutating webhook, including a Gatekeeper `Assign` mutator

For example, a Deployment without `resources` in its pod template is **not** denied by `D8RequiredResources`, because a LimitRange in the namespace may add default `requests` and `limits`. If `resources` is set partially (for example, only `limits.memory` but not `limits.cpu`), the violation is still enforced.

`D8RequiredResources` relies on an in-tree component. `D8AllowedUsers` does not: in a cluster without a matching mutator it gets no controller-level enforcement of these fields, and the violation surfaces at Pod creation instead. Pod Security Admission is not one of the sources listed above — it only validates and never modifies an object.

`D8AllowedSeccompProfiles` needs no lenient mode: a container that sets no profile in any source is allowed on a Pod as well, so controllers and Pods already behave the same.

Constraints with limited controller-level coverage

Some checks depend on data that only exists after mutation, so they cannot deny a controller even though they deny its pods.

`D8AutomountServiceAccountToken` detects the token mount either from `spec.automountServiceAccountToken: true` or from a container that mounts `/var/run/secrets/kubernetes.io/serviceaccount`. The ServiceAccount admission plugin adds that mount to the Pod, not to the controller's pod template. A workload that simply omits `automountServiceAccountToken` therefore passes the controller-level check, and the denial appears when its pods are created. To get the denial at the controller level, set `automountServiceAccountToken: false` explicitly in the pod template.

Pod validation when policies are modified or new ones are added

For all three policy categories (Pod Security Standards, operational, and security policies), there is no provision for automatically recreating existing pods when changing existing policies or adding new ones. Pods that existed prior to changes being made to the policy in use or prior to a new policy being added will continue to run until they are restarted. Upon restart, they will be validated against the new rules.

The `admission-policy-engine` module provides alerts for such cases. Each alert fires once for each policy and namespace with violating objects:

Alert	What it reports
`PodSecurityStandardsViolation`	Running pods that violate a Pod Security Standard
`OperationPolicyViolation`	Existing objects that violate an OperationPolicy
`SecurityPolicyViolation`	Existing objects that violate a SecurityPolicy
`ControllerPolicyViolation`	Pod-creating controllers that violate a policy with `enforcementAction: Deny` and are only warned about during the transition period, see Controller-level validation

The `policy` label of an alert names the policy: the name of the OperationPolicy or SecurityPolicy, or `d8-pod-security-baseline` or `d8-pod-security-restricted` for the Pod Security Standards. The `namespace` label is the namespace of the violating objects.

An alert is an ObservabilityAlert resource in the namespace of the violating objects, so the users of the namespace see it. An alert for cluster-scoped objects is a ClusterObservabilityAlert resource.

To list the alerts in a namespace, run the following command, where `<NAMESPACE>` is the namespace:

```
d8 k -n <NAMESPACE> get observabilityalerts
```

Output example:

```
NAME                                                    SEVERITY   STATUS   DURATION
SUMMARY
AGE
PodSecurityStandardsViolation-07243a74fddd8234-1792744609   3          Firing   1h
Pods in the `app` namespace violate the `d8-pod-security-baseline` Pod Security Standard.
1h
SecurityPolicyViolation-8f21cd962274a252-1792744609         3          Firing   1h
Objects in the `app` namespace violate the `restricted-app` SecurityPolicy.
1h
```

To list the alerts in every namespace and the alerts for cluster-scoped objects, run the following commands:

```
d8 k get observabilityalerts -A
d8 k get clusterobservabilityalerts
```

To view an alert, run the following command:

```
d8 k -n <NAMESPACE> get observabilityalert <ALERT_NAME> -o yaml
```

The description of the alert contains a Prometheus query that lists the violating objects and the rules they break.

Pod Security Standards

Pod Security Standards (`PSS`) is an official Kubernetes standard that defines three security levels for pods, limiting their privileges. Restrictions are enforced by prohibiting the setting of certain parameters in the pod manifest.

A layered structure is used — each higher level of protection uses all the rules of the previous level and adds its own.

The following protection levels are regulated:

- `Privileged`: An unrestricted policy with the widest possible level of permissions (no restrictions).
- `Baseline`: A minimally restrictive policy that prevents the most known and popular ways of privilege escalation. Allows using the standard (minimally specified) pod configuration.
- `Restricted`: A policy with significant restrictions. Imposes the strictest requirements on pods.

> **i** In the Deckhouse Platform, these policies are implemented using Gatekeeper and enforced by the admission controllers of the `admission-policy-engine` module, rather than the Kubernetes Pod Security Admission controller. Only the policy descriptions are taken from Kubernetes.

You can read more about each set of policies and their restrictions in the Kubernetes documentation.

Configuring PSS policies for namespaces is done by setting a special label `security.deckhouse.io/pod-policy=<POLICY_NAME>` on the corresponding namespace. The default policy can be overridden globally (in the module settings).

> **i** In namespaces named `d8-*` and `kube-*`, the `restricted` standard applies in the `warn` mode, whatever the default policy and the namespace labels. A violation is recorded in the audit and shown in the web interface Deckhouse Platform, and the workload still starts. If the module that owns the namespace has enabled policy checks for it, the standard is also applied there with the enforcement action configured for that namespace. The `security.deckhouse.io/pod-policy-action` label cannot lower the action below `warn` in these namespaces: with `dryrun`, the user still receives warnings. In such a namespace each violation is reported by both checks, so the API server response and the audit contain it twice. The module dashboard counts such a violation once.

> **i** When the `multitenancy-manager` module is enabled, it creates its own OperationPolicy objects (for example, in the `default` namespace). These are not affected by the `podSecurityStandards` settings.

Example of setting the `Restricted` policy for all pods in the `my-namespace` namespace:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
```

Additionally, it is possible to configure the policy enforcement mode. The following modes are supported:

- `deny`: Prohibit starting pods that do not satisfy the policy.
- `warn`: Start pods that do not satisfy the policy, but issue a warning.
- `dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in

security reports.

Configuring the policy enforcement mode is done by setting the label `security.deckhouse.io/pod-policy-action=<POLICY ACTION>` on the corresponding namespace. To set the policy enforcement mode globally, use the `enforcementaction` parameter.

Example of setting the `warn` mode for PSS policies for all pods in the `my-namespace` namespace:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
```

Rules of the standards

The module publishes the rules of the standards as two SecurityPolicy objects, `d8-pod-security-baseline` and `d8-pod-security-restricted`. The rules are written in the format of the SecurityPolicy parameters, the same ones you use in your own security policies. `d8-pod-security-restricted` lists every rule a `Restricted` namespace is checked against, the `Baseline` rules included. Two values in the rules have no counterpart in the SecurityPolicy description: `""` and `undefined` in `seccompProfiles.allowedProfiles` allow a container that sets no seccomp profile, and `type: ""` in `seLinux` allows a container that sets no SELinux type.

To view the rules of the `Restricted` standard, run the following command:

```
d8 k get securitypolicy d8-pod-security-restricted -o yaml
```

The `spec.policies` section lists the rules, and the `spec.controllerValidation` section shows the controller-level validation settings taken from the module configuration.

The objects describe the standards and enforce nothing by themselves. The module applies the standards to namespaces by the `security.deckhouse.io/pod-policy` label and the module settings, and `spec.match` of the objects shows the application namespaces the standard applies to with the default enforcement mode. The objects are managed by the module and cannot be modified. Configure the standards in the module settings. If a SecurityPolicy of your own is already named `d8-pod-security-baseline` or `d8-pod-security-restricted`, the module keeps your policy and does not publish that standard. The standard is still enforced. To see its rules, rename your policy.

Operational policies

Operational policies are rules aimed at achieving application security best practices, but not directly related to the validation of classic security-related parameters (for example, a list of allowed prefixes for container images, an image download policy, a list of required container images, etc.).

Operational policies are described using the `OperationPolicy` custom resource. In this resource, each parameter is responsible for a separate check applied to resources. Using the OperationPolicy custom resource allows you to define additional requirements for the resources being created (high-level declarative operational policies) without explicitly interacting with Gatekeeper.

We recommend setting the following minimum set of operational policies:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  enforcementAction: Deny
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.ru
    requiredResources:
      limits:
        - memory
      requests:
        - cpu
        - memory
    disallowedImageTags:
      - latest
    requiredProbes:
      - livenessProbe
      - readinessProbe
    maxRevisionHistoryLimit: 3
    imagePullPolicy: Always
    priorityClassNames:
    - production-high
    - production-low
    checkHostNetworkDNSPolicy: true
    checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          custom-operation-policy/enabled: "true"
```

Policy application is implemented through settings located in the `spec.match` parameter.

When specifying:

```
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          custom-operation-policy/enabled: "true"
```

To apply the above policy, it is sufficient to add the label `custom-operation-policy/enabled: "true"` to the desired namespace. Unlike `PSS`, the label name can be anything. Only a match between the label in the policy selector and the corresponding namespace is required.

You can read more detailed information about using selectors in the selector setup description.

It is also possible to specify the action to be applied for the policy. The `spec.enforcementAction` parameter is used for this. The following modes are supported:

- `Deny`: Prohibit starting pods that do not satisfy the policy.
- `Warn`: Start pods that do not satisfy the policy, but issue a warning.
- `Dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in security reports.

Based on this example, you can create your own policy with the necessary settings.

Security policies

Security policies are rules aimed at achieving application security best practices by validating the values of security-related parameters.

Security policies are described using the `SecurityPolicy` custom resource. In this resource, each parameter is responsible for a separate check applied to resources. Using this resource, it is possible to construct a security policy similar to a PSS policy of any level. Using the custom SecurityPolicy resource allows you to define additional requirements for the resources being created (high-level declarative security policies) without explicitly interacting with Gatekeeper.

Example of a security policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
    - driver: vmware
    allowedHostPorts:
    - max: 4000
      min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
    - unconfined
    allowedUnsafeSysctls:
    - kernel.*
    allowedVolumes:
    - hostPath
    - projected
    fsGroup:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - ALL
    runAsGroup:
      ranges:
      - max: 500
        min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
      - my_profile.json
      allowedProfiles:
      - Localhost
    supplementalGroups:
      ranges:
      - max: 133
        min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy: mypolicy
```

⚠ The `allowPrivilegeEscalation` and `allowPrivileged` parameters default to `false` — even if not explicitly specified. This means that containers will not be able to run in privileged mode or escalate privileges. To allow such behavior, set the parameter to `true`.

Policy application is implemented through settings located in the `spec.match` parameter.

When specifying:

```
match:
  namespaceSelector:
    labelSelector:
      matchLabels:
        security-policy: mypolicy
```

To apply the above policy, it is sufficient to add the label `security-policy: mypolicy` to the desired namespace. Unlike `PSS`, the label name can be anything. Only a match between the label in the policy selector and the corresponding namespace is required.

You can read more detailed information about using selectors in the selector setup description.

It is also possible to specify the action to be applied for the policy. The `spec.enforcementAction` parameter is used for this. The following modes are supported:

- `Deny`: Prohibit starting pods that do not satisfy the policy.
- `Warn`: Start pods that do not satisfy the policy, but issue a warning.
- `Dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in security reports.

Security policy exceptions

SecurityPolicyException is a resource that lets you create fine-grained exceptions from security policy checks for individual pods and containers. It allows you to avoid excluding an entire namespace from checks and instead define only the necessary exceptions from a specific rule for a pod or container.

Adding exceptions

To add exceptions for a pod or container, do the following:

1. Create a SecurityPolicyException object describing the required exceptions.

   It is recommended that you describe the reason for each exception in the rule's `metadata` field (for example, `metadata.description`). This makes auditing and maintenance easier.

2. In the pod template (usually via `spec.template.metadata.labels` in a Deployment, StatefulSet, or DaemonSet resource), add one of the following labels referencing the exception:

   - `security.deckhouse.io/security-policy-exception: <exception-name>`: Exception for the entire pod.
   - `security.deckhouse.io/security-policy-exception.container.<container-name>: <exception-name>`: Exception for a specific container.

Priority when selecting an exception for a container:

1. The label `security.deckhouse.io/security-policy-exception.container.<container-name>` is checked first.
2. If the container-specific label is absent, the exception from `security.deckhouse.io/security-policy-exception` is used.

> ⚠ If a container-specific label is set for a container but it points to an invalid or non-existent SecurityPolicyException object, it still has priority over the global label and may lead to pod placement denial.

Configuration example

For this example, consider a pod that requires:

- Permission to use the `hostNetwork` parameter for the entire pod.
- Permission to use the `privileged` parameter only for the `sample-init` container.

Without the SecurityPolicyException resource, allowing these parameters would require implementing a custom security policy where these settings could be allowed for any pod in the cluster.

With SecurityPolicyException, it is enough to create only the following resources:

- Exception to allow the `hostNetwork` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicyException
metadata:
  name: allow-hostnetwork-pod
spec:
  network:
    hostNetwork:
      allowedValue: true
      metadata:
        description: >-
          Pod requires host network mode for node-level network diagnostics.
```

- Exception to allow the `privileged` parameter in the `sample-init` container:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicyException
metadata:
  name: allow-privileged-init-container
spec:
  securityContext:
    privileged:
      allowedValue: true
      metadata:
        description: >-
          Container init requires privileged mode to access host-level networking
features.
```

After that, the corresponding labels need to be added to the pod template:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example
spec:
  template:
    metadata:
      labels:
        # General exception applicable to the entire pod.
        security.deckhouse.io/security-policy-exception: allow-hostnetwork-pod
        # Exception applicable to the sample-init container.
        security.deckhouse.io/security-policy-exception.container.sample-init: allow-
privileged-init-container
    spec:
      hostNetwork: true
    ...
    containers:
      - name: sample-init
        securityContext:
          privileged: true
```

Modifying Kubernetes resources

The module allows you to use the Gatekeeper Custom Resources to modify objects in the cluster, such as:

- AssignMetadata — defines changes to the `metadata` section of a resource.
- Assign — any change outside the `metadata` section.
- ModifySet — adds or removes entries from a list, such as the arguments to a container.
- AssignImage — to change the `image` parameter of the resource.

You can read more about the available options in the gatekeeper documentation.

Availability of the module components

The module is on the critical path of the cluster: while its validating webhook is unavailable, the API server rejects the requests the webhook intercepts.

Why the validating webhook is a critical component

The `gatekeeper-controller-manager` deployment serves the ValidatingWebhookConfiguration named `d8-admission-policy-engine-config`. Its main webhook is configured with `failurePolicy: Fail`, which means that a request the API server cannot deliver to the webhook is rejected rather than admitted.

While the webhook is unavailable, no object bypasses the policies, but the objects the webhook intercepts cannot be created or changed during that time either.

While no replica of `gatekeeper-controller-manager` is available, the following stops working in the namespaces the webhook covers:

- Operations on the objects listed in Controller-level validation, including `d8 k exec` and `d8 k attach`, as well as `d8 k debug`
- Creation, modification and deletion of Role, RoleBinding and Gatekeeper constraints

The mutating webhook is configured differently: its `failurePolicy` is `Ignore`, and an unavailable deployment only means that mutations are not applied.

Disabling the module unblocks the cluster, but the control goes with it: no policy is enforced any more, and an object a policy used to forbid is created without hindrance. The FAQ describes what to do while the webhook is unavailable.

Which objects are excluded from validation

The exclusions define which objects can be created and changed while the webhook is unavailable.

The webhooks of the module do not validate the following:

- Objects in namespaces with the `heritage: deckhouse` label, which the main webhook excludes through `namespaceSelector`
- Objects with the `gatekeeper.sh/operation: webhook` label that Gatekeeper sets on its own pods, which the webhooks exclude through `objectSelector`
- Requests from service accounts of the `d8-virtualization` namespace

Namespaces that also carry the `security.deckhouse.io/enable-security-policy-check` label, including `d8-admission-policy-engine`, are validated by a separate webhook.

The exclusion by the `gatekeeper.sh/operation: webhook` label is required in the namespace of the module. Without this exclusion, the webhook could not recover from its own outage. Once the last replica is gone, the API server would reject the creation of a replacement pod, since no replica is available to validate it.

`d8 k exec` and `d8 k attach` are not covered by the exclusion by label. The object of such a request cannot have labels, and the API server never calls a webhook with `objectSelector` for it. These requests are intercepted by a separate webhook without `objectSelector`, which excludes the same namespaces as the main webhook. In the `d8-*` and `kube-*` namespaces they do not reach Gatekeeper at all.

Third-party components

List of third-party software used in the `admission-policy-engine` module:

- Gatekeeper 3.23.1

  License: Apache License 2.0

  Policy-based control for cloud native environments

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** Available with limitations in: **Open/CE, Core, BE, SE, SE+**
>
> Available without limitations in: **Ultimate/EE**

> **i** Included in extensions: **Advanced Infrastructure Security**

> **i** The module lifecycle stage: General Availability

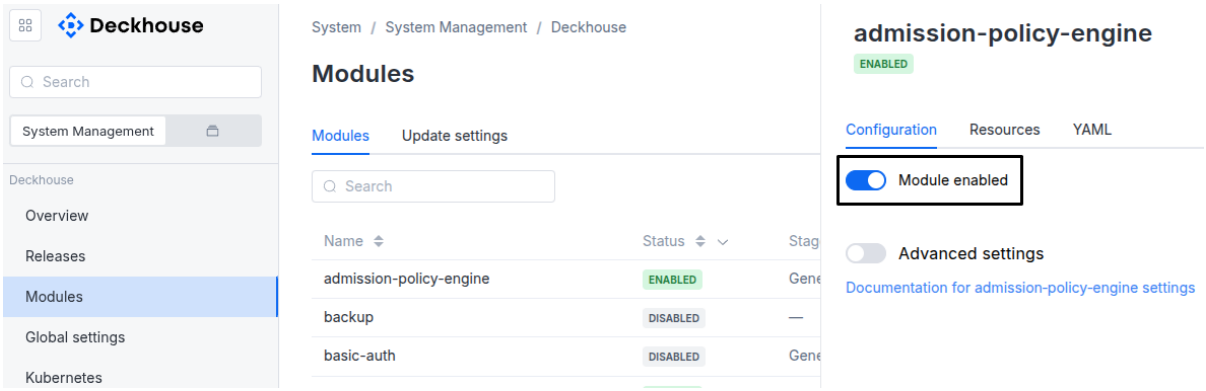
The module has 9 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `admission-policy-engine` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable admission-policy-engine
```

- **Using ModuleConfig `admission-policy-engine`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `admission-policy-engine` (create it if necessary);

  Example of a manifest to enable module `admission-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  enabled: true
```

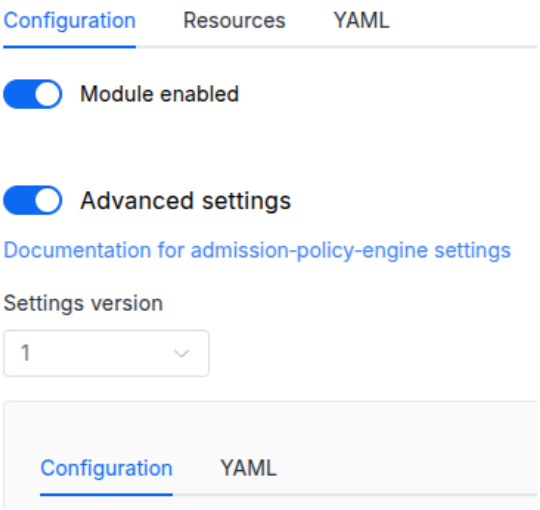
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `admission-policy-engine` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `admission-policy-engine` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `admission-policy-engine`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `admission-policy-engine` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc admission-policy-engine
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `admission-policy-engine` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `admission-policy-engine`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: admission-policy-engine
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.denyVulnerableImages**
    object

    Available in editions: Ultimate/EE

    Trivy provider settings.

    The Trivy provider prevents creation of the Pod, Deployment, StatefulSet, and DaemonSet resources with images containing vulnerabilities in namespaces labeled with `security.deckhouse.io/trivy-provider: ""`.

    Images of every container are checked, including init containers and ephemeral containers. As a result, attaching an ephemeral container with `kubectl debug` is denied if its image contains vulnerabilities of a disallowed severity level.

    Deprecated. These settings have been moved to the `operator-trivy` module. Migrate your

configuration.

- **settings.denyVulnerableImages.allowedSeverityLevels**
  array of strings

  List of allowed vulnerability severity levels.

  Images containing only vulnerabilities of the specified severity levels will not be blocked.

  - **Element of the array**
    string

    Allowed values: `UNKNOWN`, `LOW`, `MEDIUM`, `HIGH`, `CRITICAL`

- **settings.denyVulnerableImages.enabled**
  boolean

  Enable the Trivy provider.

  Default: `false`

- **settings.denyVulnerableImages.registrySecrets**
  array of objects

  List of additional Secrets for accessing private registries.

  By default, the `deckhouse-registry` Secret is used to download images for scanning.

  Default: `[]`

  - **settings.denyVulnerableImages.registrySecrets.name**
    string

    Required value

  - **settings.denyVulnerableImages.registrySecrets.namespace**
    string

    Required value

- **settings.denyVulnerableImages.storageClass**
  string

  Name of the StorageClass used by the Trivy provider.

  If the value is not specified, the StorageClass set in the global `settings.modules.storageClass` parameter is used instead.

  The global `settings.modules.storageClass` parameter is only applied when the module is enabled. Changing the global parameter while the module is enabled will not trigger disk re-provisioning.

  > **i** **Warning.** If you specify a value different from the current StorageClass used in the existing PersistentVolumeClaim, the disk will be re-provisioned, and all data will be deleted.

  If you set it to `false`, `emptyDir` will be forcibly used instead of a PVC.

  Examples:

  ```
  storageClass: ceph-ssd
  ```

  ```
  storageClass: 'false'
  ```

- **settings.imageSignatureVerification**
  object

  Available in editions: SE+, Ultimate/EE

  Settings of the image signature verification configured by the `verifyImageSignatures` policy of the SecurityPolicy resource.

  - **settings.imageSignatureVerification.registryCredentialsCacheTTLSeconds**
    integer

    Time in seconds for which the credentials of a private registry stay cached after being read from the `dockerCfg` field of a SecurityPolicy.

    After you change `dockerCfg`, the signature verification keeps using the previous credentials until the cache entry expires. A lower value applies the new credentials sooner, at the cost of more frequent reads of the Secret from the Kubernetes API.

Default: `300`

Allowed values: `30 <= X <= 43200`

Examples:

```
registryCredentialsCacheTTLSeconds: 300
```

```
registryCredentialsCacheTTLSeconds: 3600
```

- **settings.podSecurityStandards**
  object

  Pod Security Standards policy settings (PSS).

  - **settings.podSecurityStandards.controllerValidation**
    boolean

    Applies constraints to pod-creating controllers (Deployment, StatefulSet, DaemonSet, Job, CronJob, ReplicationController) in addition to Pods.

    Set to `true` (default) to validate both Pods and controllers, or to `false` to validate only Pods and leave controllers unchecked.

    How a violation in a controller is reported is set by controllerValidationEnforcement.

    The parameter is the default for every constraint that reads a pod spec, including the ones generated from OperationPolicy and SecurityPolicy resources. A policy overrides it with its own `spec.controllerValidation` section.

    Default: `true`

  - **settings.podSecurityStandards.controllerValidationEnforcement**
    string

    Action on a pod-creating controller that violates a policy with `enforcementAction: Deny`.

    - `Warn` — the controller is accepted with a warning, and the Pods it creates are denied. The value is the default for a transition period, while workloads are brought in line with the controller-level checks.
    - `Policy` — the controller is checked with the action of the policy, so a denying policy denies the controller as well. The value is to become the default after the transition period.

    A policy with another action keeps it for controllers: `Warn` warns and `Dryrun` only records the violation.

    The parameter applies to the Pod Security Standards and is the default for OperationPolicy and SecurityPolicy resources. A policy overrides it with `spec.controllerValidation.mode`. The parameter takes effect when controllerValidation is `true`.

    Default: `Warn`

    Allowed values: `Warn`, `Policy`

  - **settings.podSecurityStandards.controllerValidationExcludedPolicies**
    array of strings

    Pod Security Standards rules that are checked on Pods only.

    The rules are named as the SecurityPolicy parameters the standards are published with, in the `d8-pod-security-baseline` and `d8-pod-security-restricted` objects. An excluded rule is checked on Pods only in both standards. The parameter takes effect when controllerValidation is `true`.

    Some rules are checked by one constraint and are excluded together: listing any rule of a group checks the whole group on Pods only. The groups are `allowHostPID` with `allowHostIPC`, and `allowHostNetwork` with `allowedHostPorts`.

    Default: `[]`

    Example:

    ```
    controllerValidationExcludedPolicies:
    - allowPrivilegeEscalation
    - requiredDropCapabilities
    ```

    - **Element of the array**
      string

      Allowed values: `allowPrivileged`, `allowPrivilegeEscalation`, `allowHostPID`, `allowHostIPC`, `allowHostNetwork`, `allowedHostPorts`, `allowedAppArmor`,

`allowedCapabilities`, `requiredDropCapabilities`, `allowedHostPaths`, `allowedProcMount`, `seLinux`, `allowedUnsafeSysctls`, `seccompProfiles`, `allowedVolumes`, `runAsUser`

- **settings.podSecurityStandards.defaultPolicy**
  string

  Sets the default Pod Security Standards policy for all **non-system** namespaces:

  - `Privileged` — an unrestricted policy. Privilege escalation is possible with this policy;
  - `Baseline` — a policy with minimum restrictions that prevents privilege escalation;
  - `Restricted` — a policy with maximum restrictions that conforms to current best practices for securely running applications in a cluster.

  Default: `Baseline`

  Allowed values: `Privileged`, `Baseline`, `Restricted`

- **settings.podSecurityStandards.enforcementAction**
  string

  The enforcement action to control what to do with the result of the constraint.

  - `Deny` — Deny action.
  - `Dryrun` — No action. It is used when debugging. Information about the event can be viewed in the web-interface Deckhouse Platform or with `d8 k`.
  - `Warn` — Same as `Dryrun`. In addition to the event information, it provides some info on why that constraint would have been denied if you had set `Deny` instead of `Warn`.

  Applies to application namespaces. Namespaces named `d8-*` and `kube-*` are governed by the module itself: a violation is reported there and, unless the module that owns the namespace has opted it into enforcement, the workload still starts.

  Default: `Deny`

  Allowed values: `Warn`, `Deny`, `Dryrun`

- **settings.podSecurityStandards.policies**
  object

  Sets additional policy parameters.

  - **settings.podSecurityStandards.policies.hostPorts**
    object

    HostPort constraint settings.

    - **settings.podSecurityStandards.policies.hostPorts.knownRanges**
      array of objects

      Set the range of known ports which will be allowed in a hostPort binding.

      - **settings.podSecurityStandards.policies.hostPorts.knownRanges.max**
        integer
      - **settings.podSecurityStandards.policies.hostPorts.knownRanges.min**
        integer

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

OperationPolicy

Scope: Cluster
Version: v1alpha1

Describes an operation policy for a cluster.

Each CustomResource `OperationPolicy` describes rules for objects in a cluster.

- **spec**
  object

  Required value

  - **spec.controllerValidation**
    object

    Validation of the pod-creating controllers against the policy.

    The controllers are Deployment, StatefulSet, DaemonSet, Job, CronJob and ReplicationController. A controller is checked when it is created or updated, so a violation is reported before any Pod is created. Pods are checked in any case.

    The section overrides the module parameters podSecurityStandards.controllerValidation and podSecurityStandards.controllerValidationEnforcement for this policy.

    - **spec.controllerValidation.excludedPolicies**
      array of strings

      Rules of the policy that are checked on Pods only.

      Use it for a rule that a mutating webhook satisfies on the Pod but not in the controller's pod template, for example `requiredResources` with the Vertical Pod Autoscaler.

      Example:

      ```
      excludedPolicies:
      - requiredResources
      ```

      - **Element of the array**
        string

        Allowed values: `allowedRepos`, `requiredResources`, `disallowedImageTags`, `requiredProbes`, `imagePullPolicy`, `priorityClassNames`, `checkHostNetworkDNSPolicy`, `checkContainerDuplicates`, `disallowedTolerations`, `gpuResourceRestriction`

    - **spec.controllerValidation.mode**
      string

      Controller validation mode of the policy:

      - `Enforce` — controllers are checked against every rule of the policy except the ones listed in excludedPolicies, with the action of the policy;
      - `Warn` — controllers are checked in the same way, but a policy with `enforcementAction: Deny` only warns about a violation in a controller. The Pods the controller creates are denied;
      - `Disabled` — only Pods are checked.

      If the parameter is omitted, the module parameters podSecurityStandards.controllerValidation and podSecurityStandards.controllerValidationEnforcement are used.

      Allowed values: `Enforce`, `Warn`, `Disabled`

  - **spec.enforcementAction**
    string

    The enforcement action to control what to do with the result of the constraint.

    - `Deny` — Deny action.
    - `Dryrun` — No action. It is used when debugging. Information about the event can be viewed in Deckhouse Console.
    - `Warn` — Same as `Dryrun`. In addition to the event information, it provides some info on why that constraint would have been denied if you had set `Deny` instead of `Warn`.

    `Deny` holds in application namespaces. In a namespace named `d8-*` or `kube-*` the policy always warns, whatever action is set here.

    Default: `Deny`

    Allowed values: `Warn`, `Deny`, `Dryrun`

  - **spec.match**
    object

    Required value

    - **spec.match.labelSelector**
      object

      Specifies the label selector to filter Pods with.

You can get more into here.

- **spec.match.labelSelector.matchExpressions**
  array of objects

  List of label expressions for Pods.

  Example:

  ```
  matchExpressions:
  - key: tier
    operator: In
    values:
    - production
    - staging
  ```

  - **spec.match.labelSelector.matchExpressions.key**
    string

    Required value

  - **spec.match.labelSelector.matchExpressions.operator**
    string

    Required value

    Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

  - **spec.match.labelSelector.matchExpressions.values**
    array of strings
- **spec.match.labelSelector.matchLabels**
  object

  List of labels which Pod should have.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.match.namespaceSelector**
  object

  Specifies the namespace selector to filter objects with.

  - **spec.match.namespaceSelector.excludeNames**
    array of strings

    Include all namespaces except a particular set. Support glob pattern.

  - **spec.match.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces.

    You can get more info in the documentation.

    - **spec.match.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      List of label expressions for namespaces.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      ```

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.key**
        string

        Required value

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**
        string

Required value

Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.namespaceSelector.labelSelector.matchExpressions.values**
  array of strings
- **spec.match.namespaceSelector.labelSelector.matchLabels**
  object

  List of labels which a namespace should have.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.match.namespaceSelector.matchNames**
  array of strings

  Include only a particular set of namespaces. Supports glob pattern.

- **spec.policies**
  object

  Required value

  - **spec.policies.allowedRepos**
    array of strings

    The list of prefixes a container image is allowed to have.

    - **Element of the array**
      string

      Example:

      ```
      registry.deckhouse.io
      ```

  - **spec.policies.checkContainerDuplicates**
    boolean

    Check container names and env variables for duplicates.

  - **spec.policies.checkHostNetworkDNSPolicy**
    boolean

    Check `ClusterFirstWithHostNet` dnsPolicy is set for Pods with `hostNetwork: true`.

  - **spec.policies.disallowedImageTags**
    array of strings

    Requires container images to have an image tag different from the ones in the specified list.

    Example:

    ```
    disallowedImageTags: latest
    ```

  - **spec.policies.disallowedTolerations**
    array of objects

    Deny Pods that have tolerations from the ones in the specified list.

    See more in the Kubernetes documentation.

    Each item in the list is a toleration pattern matched against the `key`, `operator`, `value`, and `effect` fields. A partial match on these fields is enough to consider it a match.

    - **spec.policies.disallowedTolerations.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **spec.policies.disallowedTolerations.key**
      string
    - **spec.policies.disallowedTolerations.operator**
      string

      Allowed values: `Exists`, `Equal`

    - **spec.policies.disallowedTolerations.value**
      string

- **spec.policies.gpuResourceRestriction**
  object

  Restricts GPU resource usage in pods.

  Pods that request GPU resources are denied unless the namespace has the specified label. The same check runs on the pod-creating controllers, so a controller whose pod template requests GPU resources is denied on creation and update. Both `resources.requests` and `resources.limits` of every container, init container, and ephemeral container are inspected.

  The namespace label is read from the Gatekeeper cache. While the namespace is missing from that cache, for example when a namespace and a pod are applied together and the namespace has not been cached yet, the policy denies the pods that request GPU resources in that namespace.

  - **spec.policies.gpuResourceRestriction.gpuResourcePatterns**
    array of strings

    List of regular expressions matched against GPU resource names in pod specs.

    A resource name that matches any of these patterns is considered a GPU resource. Patterns are not anchored implicitly: `nvidia.com/gpu` also matches the `not-nvidia.com/gpu` resource name. Use `^` and `$` to match the whole resource name.

    Default: `["^nvidia\\.com/.*$"]`

    Example:

    ```
    gpuResourcePatterns:
    - "^nvidia\\.com/.*$"
    - "^amd\\.com/gpu$"
    - "^intel\\.com/gpu$"
    ```

  - **spec.policies.gpuResourceRestriction.namespaceLabel**
    object

    Label that must be present on the namespace to allow GPU resource usage.

    - **spec.policies.gpuResourceRestriction.namespaceLabel.key**
      string

      Label key that must be present on the namespace.

      Default: `gpu.deckhouse.io/enabled`

    - **spec.policies.gpuResourceRestriction.namespaceLabel.value**
      string

      Expected label value.

      Default: `true`

- **spec.policies.imagePullPolicy**
  string

  Required image pull policy for containers.

  Allowed values: `Always`, `IfNotPresent`

- **spec.policies.ingressClassNames**
  array of strings

  List of allowed ingress class names.

- **spec.policies.maxRevisionHistoryLimit**
  integer

  A maximum value for a revision history.

- **spec.policies.priorityClassNames**
  array of strings

  List of allowed priority class names.

- **spec.policies.replicaLimits**
  object

  A range of allowed replicas. Values are inclusive.

  - **spec.policies.replicaLimits.maxReplicas**
    integer

    The maximum number of replicas allowed, inclusive.

- **spec.policies.replicaLimits.minReplicas**
  integer

  The minimum number of replicas allowed, inclusive.

- **spec.policies.requiredAnnotations**
  object

  A list of annotations and values the object must specify.

  - **spec.policies.requiredAnnotations.annotations**
    array of objects
    - **spec.policies.requiredAnnotations.annotations.allowedRegex**
      string

      If specified, a regular expression, the annotation’s value must match. The value must contain at least one match for the regular expression.

    - **spec.policies.requiredAnnotations.annotations.key**
      string

      The required annotation.

  - **spec.policies.requiredAnnotations.watchKinds**
    array of strings

    The list of Kubernetes objects in the format `$apiGroup/$kind` to watch the annotations on.

    - **Element of the array**
      string

      Pattern: `^[a-z]*/[a-zA-Z]+$`

      Examples:

      ```
      apps/Deployment
      ```

      ```
      "/Pod"
      ```

      ```
      networking.k8s.io/Ingress
      ```

- **spec.policies.requiredLabels**
  object

  A list of labels and values the object must specify.

  - **spec.policies.requiredLabels.labels**
    array of objects
    - **spec.policies.requiredLabels.labels.allowedRegex**
      string

      If specified, a regular expression, the label’s value must match. The value must contain at least one match for the regular expression.

    - **spec.policies.requiredLabels.labels.key**
      string

      The required label.

  - **spec.policies.requiredLabels.watchKinds**
    array of strings

    The list of Kubernetes objects in the format `$apiGroup/$kind` to watch the labels on.

    - **Element of the array**
      string

      Pattern: `^[a-z]*/[a-zA-Z]+$`

      Examples:

      ```
      apps/Deployment
      ```

      ```
      "/Pod"
      ```

      ```
      networking.k8s.io/Ingress
      ```

- **spec.policies.requiredProbes**
  array of strings

The list of probes that are required (e.g. `readinessProbe` and `livenessProbe`)

Example:

```
requiredProbes:
- readinessProbe
- livenessProbe
```

- **Element of the array**
  string

  Allowed values: `livenessProbe`, `readinessProbe`, `startupProbe`

- **spec.policies.requiredResources**
  object

  Requires containers to have defined resources set.

  - **spec.policies.requiredResources.limits**
    array of strings

    A list of limits that should be enforced (CPU, memory, or both).

    - **Element of the array**
      string

      Allowed values: `cpu`, `memory`

  - **spec.policies.requiredResources.requests**
    array of strings

    A list of requests that should be enforced (CPU, memory, or both).

    - **Element of the array**
      string

      Allowed values: `cpu`, `memory`

- **spec.policies.storageClassNames**
  array of strings

  List of allowed storage class names.

SecurityPolicyException

Short names: spe

Scope: Namespaced
Version: v1alpha1

Describes exceptions to the cluster's security policies. Each SecurityPolicyException resource defines a set of specific allowances.

- **status**
  object
  - **status.deckhouse**
    object
    - **status.deckhouse.observed**
      object

      Contains last timestamp when the resource change was noted by the operator and its checksum.

      - **status.deckhouse.observed.checkSum**
        string

        The checksum of last observed resource.

      - **status.deckhouse.observed.lastTimestamp**
        string

        A timestamp of when the operator last noted a change to the resource.

    - **status.deckhouse.processed**
      object

      Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

      - **status.deckhouse.processed.checkSum**
        string

        The checksum of last applied resource.

        - **status.deckhouse.processed.lastTimestamp**
          string

          A timestamp of when the resource was last applied in the cluster.

      - **status.deckhouse.synced**
        string

        True if last observed version of the resource was successfully applied in the cluster.

- **spec**
  object

  Required value

  - **spec.network**
    object

    Exceptions for network-related parameters.

    - **spec.network.hostIPC**
      object

      Exceptions for the `hostIPC` parameter.

      - **spec.network.hostIPC.allowedValue**
        boolean

        Required value

        Allowed value for `hostIPC`.

        Default: `false`

      - **spec.network.hostIPC.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostNetwork**
      object

      Exceptions for the `hostNetwork` parameter. For pods with `hostNetwork: true`, an additional exception is performed on the ports specified in `containerPort`.

      - **spec.network.hostNetwork.allowedValue**
        boolean

        Required value

        Allowed value for `hostNetwork`.

        Default: `false`

      - **spec.network.hostNetwork.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostPID**
      object

      Exceptions for the `hostPID` parameter.

      - **spec.network.hostPID.allowedValue**
        boolean

        Required value

        Allowed value for `hostPID`.

        Default: `false`

      - **spec.network.hostPID.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostPorts**
      array of objects

      Exceptions for host ports.

      - **spec.network.hostPorts.metadata**
        object

Metadata related to the exception.

- **spec.network.hostPorts.port**
  integer

  Allowed host port.

- **spec.network.hostPorts.protocol**
  string

  Protocol for the host port.

  Allowed values: `TCP`, `UDP`

- **spec.securityContext**
  object

  Exceptions for parameters in the `securityContext` section.

  - **spec.securityContext.allowPrivilegeEscalation**
    object

    Exceptions for the `allowPrivilegeEscalation` parameter.

    - **spec.securityContext.allowPrivilegeEscalation.allowedValue**
      boolean

      Required value

      Allowed value for `allowPrivilegeEscalation`.

      Default: `false`

    - **spec.securityContext.allowPrivilegeEscalation.metadata**
      object

      Metadata related to the exception.

  - **spec.securityContext.appArmorProfile**
    object

    List of allowed AppArmor profile values for pods and containers.

    Supported formats:

    - Via annotations: `runtime/default`, `docker/default`, `unconfined`, `localhost/some-profile.json`. `localhost/*` allows any local profile.
    - Via `securityContext`: `RuntimeDefault`, `Unconfined`, `Localhost`. For `Localhost`, specify allowed profiles using the `allowedLocalhostFiles` parameter.

    Profile types:

    - `Unconfined`: No restrictions (not recommended for security reasons).
    - `RuntimeDefault`: The default profile provided by the container runtime (such as Docker or CRI-O).
    - `Localhost`: A custom profile defined on the host and tailored for the application.

    Using `*` allows all profiles. It's not necessary to specify both formats as they are automatically mapped to each other.

    - **spec.securityContext.appArmorProfile.allowedValues**
      array of strings

      Required value

      Allowed values:

      - `runtime/default`
      - `unconfined`
      - `localhost/*` (where `*` may be a profile name or a wildcard)
      - **Element of the array**
        string

        Pattern: `^(runtime\/default|unconfined|localhost\/.+)$`

    - **spec.securityContext.appArmorProfile.metadata**
      object

      Metadata related to the exception.

  - **spec.securityContext.capabilities**
    object

    Exceptions for the `capabilities` parameter.

    - **spec.securityContext.capabilities.allowedValues**

object

Required value

Allowed values for adding or dropping Linux capabilities.

- **spec.securityContext.capabilities.allowedValues.add**
  array of strings

  List of capabilities that are allowed to be added to the container.

  Default: `[]`

- **spec.securityContext.capabilities.allowedValues.drop**
  array of strings

  List of capabilities that are allowed to be dropped from the container.

  Default: `["all"]`

- **spec.securityContext.capabilities.metadata**
  object

  Metadata related to the exception.

- **spec.securityContext.privileged**
  object

  Exceptions for the `privileged` parameter.

  - **spec.securityContext.privileged.allowedValue**
    boolean

    Required value

    Allowed value for `privileged`.

    Default: `false`

  - **spec.securityContext.privileged.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.procMount**
  object

  Exceptions for the `procMount` parameter.

  - **spec.securityContext.procMount.allowedValues**
    array of strings

    Required value

    List of allowed `procMount` values.

    - **Element of the array**
      string

      Allowed values: `Default`, `Unmasked`

  - **spec.securityContext.procMount.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.readOnlyRootFilesystem**
  object

  Exceptions for the `readOnlyRootFilesystem` parameter.

  - **spec.securityContext.readOnlyRootFilesystem.allowedValue**
    boolean

    Required value

    Allowed value for `readOnlyRootFilesystem`.

    Default: `false`

  - **spec.securityContext.readOnlyRootFilesystem.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.runAsNonRoot**
  object

Exceptions for the `runAsNonRoot` parameter.

- **spec.securityContext.runAsNonRoot.allowedValue**
  boolean

  Required value

  Allowed value for `runAsNonRoot`.

  Default: `true`

- **spec.securityContext.runAsNonRoot.metadata**
  object

  Metadata related to the exception.

- **spec.securityContext.runAsUser**
  object

  Exceptions for the `runAsUser` parameter.

  - **spec.securityContext.runAsUser.allowedValues**
    array of integers

    Required value

    List of allowed user IDs (UIDs).

  - **spec.securityContext.runAsUser.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.seLinuxOptions**
  object

  Exceptions for the `seLinuxOptions` parameter.

  - **spec.securityContext.seLinuxOptions.allowedValues**
    array of objects

    Required value

    List of allowed SELinux configurations.

    - **spec.securityContext.seLinuxOptions.allowedValues.level**
      string

      SELinux level label.

    - **spec.securityContext.seLinuxOptions.allowedValues.role**
      string

      SELinux role label.

    - **spec.securityContext.seLinuxOptions.allowedValues.type**
      string

      SELinux type label.

    - **spec.securityContext.seLinuxOptions.allowedValues.user**
      string

      SELinux user label.

  - **spec.securityContext.seLinuxOptions.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.seccompProfile**
  object

  Exceptions for the `seccompProfile` parameter.

  - **spec.securityContext.seccompProfile.allowedValues**
    array of strings

    Required value

    List of allowed seccomp profiles.

    A value that ends with `*` allows every profile that starts with the part before `*`. For example, `localhost/profiles/*` allows `localhost/profiles/app.json` and `localhost/profiles/team/app.json`.

- **spec.securityContext.seccompProfile.metadata**
  object

  Metadata related to the exception.

- **spec.securityContext.sysctls**
  object

  Exceptions for the `sysctls` parameter.

  - **spec.securityContext.sysctls.allowedValues**
    array of objects

    Required value

    List of allowed `sysctl` settings.

    - **spec.securityContext.sysctls.allowedValues.name**
      string

      Name of the `sysctl` parameter.

      A name that ends with `*` allows every `sysctl` parameter that starts with the part before `*`. For example, `net.core.*` allows `net.core.rmem_max` and `net.core.somaxconn`.

    - **spec.securityContext.sysctls.allowedValues.value**
      string

      Allowed value for the specified `sysctl` parameter.

  - **spec.securityContext.sysctls.metadata**
    object

    Metadata related to the exception.

- **spec.volumes**
  object

  Exceptions for volume-related parameters.

  - **spec.volumes.hostPath**
    object

    Exceptions for `hostPath` volumes.

    - **spec.volumes.hostPath.allowedValues**
      array of objects

      Required value

      List of allowed `hostPath` configurations.

      - **spec.volumes.hostPath.allowedValues.metadata**
        object

        Metadata related to the exception.

        - **spec.volumes.hostPath.allowedValues.metadata.description**
          string

          Description of the exception.

      - **spec.volumes.hostPath.allowedValues.path**
        string

        Allowed `hostPath`.

      - **spec.volumes.hostPath.allowedValues.pathType**
        string

        How the path parameter is matched against the `hostPath` of a volume.

        The values follow the path types of an Ingress:

        - `Exact` — the exception allows only the exact path;
        - `Prefix` — the exception allows the path and every path nested under it. Paths are compared by their components, and trailing slashes are ignored. For example, `/var/lib` allows `/var/lib`, `/var/lib/` and `/var/lib/docker`, but does not allow `/var/libvirt` or `/var`.

        The readOnly parameter applies to nested paths the same way as to the path itself.

        Default: `Exact`

        Allowed values: `Exact`, `Prefix`

Example:

```
pathType: Prefix
```

- **spec.volumes.hostPath.allowedValues.readOnly**
  boolean

  A `readOnly` value for the `hostPath`.

  Default: `false`

- **spec.volumes.types**
  object

  Exceptions for volume types.

  - **spec.volumes.types.allowedValues**
    array of strings

    Required value

    List of allowed volume types.

  - **spec.volumes.types.metadata**
    object

    Metadata related to the exception.

SecurityPolicy

Scope: Cluster
Version: v1alpha1

Describes a security policy for a cluster.

Each `SecurityPolicy` custom resource describes rules for the objects in the cluster.

- **status**
  object
  - **status.deckhouse**
    object
    - **status.deckhouse.observed**
      object

      Contains last timestamp when the resource change was noted by the operator and its checksum.

      - **status.deckhouse.observed.checkSum**
        string

        The checksum of last observed resource.

      - **status.deckhouse.observed.lastTimestamp**
        string

        A timestamp of when the operator last noted a change to the resource.

    - **status.deckhouse.processed**
      object

      Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

      - **status.deckhouse.processed.checkSum**
        string

        The checksum of last applied resource.

      - **status.deckhouse.processed.lastTimestamp**
        string

        A timestamp of when the resource was last applied in the cluster.

    - **status.deckhouse.synced**
      string

      True if last observed version of the resource was successfully applied in the cluster.

- **spec**
  object

  Required value

  - **spec.controllerValidation**
    object

Validation of the pod-creating controllers against the policy.

The controllers are Deployment, StatefulSet, DaemonSet, Job, CronJob and ReplicationController. A controller is checked when it is created or updated, so a violation is reported before any Pod is created. Pods are checked in any case.

The section overrides the module parameters podSecurityStandards.controllerValidation and podSecurityStandards.controllerValidationEnforcement for this policy.

- **spec.controllerValidation.excludedPolicies**
  array of strings

  Rules of the policy that are checked on Pods only.

  Use it for a rule that a mutating webhook satisfies on the Pod but not in the controller's pod template, for example `requiredDropCapabilities` when the webhook adds `drop: [ALL]` to the containers.

  Some rules are checked by one constraint and are excluded together: listing any rule of a group checks the whole group on Pods only. The groups are:

  - `allowHostPID` and `allowHostIPC`;
  - `allowHostNetwork` and `allowedHostPorts`;
  - `fsGroup`, `runAsUser`, `runAsGroup` and `supplementalGroups`;
  - `allowedUnsafeSysctls` and `forbiddenSysctls`.

  `allowedCapabilities` and `requiredDropCapabilities` are excluded one by one.

  Example:

  ```
  excludedPolicies:
  - requiredDropCapabilities
  ```

  - **Element of the array**
    string

    Allowed values: `allowPrivileged`, `allowPrivilegeEscalation`, `allowHostPID`, `allowHostIPC`, `allowHostNetwork`, `allowedHostPorts`, `readOnlyRootFilesystem`, `automountServiceAccountToken`, `allowedFlexVolumes`, `allowedVolumes`, `allowedHostPaths`, `allowedCapabilities`, `requiredDropCapabilities`, `allowedAppArmor`, `allowedProcMount`, `fsGroup`, `runAsUser`, `runAsGroup`, `supplementalGroups`, `seLinux`, `allowedUnsafeSysctls`, `forbiddenSysctls`, `seccompProfiles`, `verifyImageSignatures`

- **spec.controllerValidation.mode**
  string

  Controller validation mode of the policy:

  - `Enforce` — controllers are checked against every rule of the policy except the ones listed in excludedPolicies, with the action of the policy;
  - `Warn` — controllers are checked in the same way, but a policy with `enforcementAction: Deny` only warns about a violation in a controller. The Pods the controller creates are denied;
  - `Disabled` — only Pods are checked.

  If the parameter is omitted, the module parameters podSecurityStandards.controllerValidation and podSecurityStandards.controllerValidationEnforcement are used.

  Allowed values: `Enforce`, `Warn`, `Disabled`

- **spec.enforcementAction**
  string

  An enforcement action as a result of the constraint:

  - `Deny` — Deny action.
  - `Dryrun` — No action. Used for debugging. Information about the event can be viewed in Deckhouse Console.
  - `Warn` — No action; similar to `Dryrun`. Provides information about the constraint that would result in a denial if the `Deny` action is used.

  `Deny` holds in application namespaces. In a namespace named `d8-*` or `kube-*` the policy always warns, whatever action is set here.

  Default: `Deny`

  Allowed values: `Warn`, `Deny`, `Dryrun`

- **spec.match**
  object

  Required value

Container filtering rules. Use selectors to specify the pods and containers to which you want to apply the policy.

- **spec.match.labelSelector**
  object

  Specifies the label selector to filter Pods with.

  You can get more into here.

  - **spec.match.labelSelector.matchExpressions**
    array of objects

    The list of label expressions for Pods.

    Example:

    ```
    matchExpressions:
    - key: tier
      operator: In
      values:
      - production
      - staging
    ```

    - **spec.match.labelSelector.matchExpressions.key**
      string

      Required value

    - **spec.match.labelSelector.matchExpressions.operator**
      string

      Required value

      Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

    - **spec.match.labelSelector.matchExpressions.values**
      array of strings
  - **spec.match.labelSelector.matchLabels**
    object

    The list of the labels that the Pod should have.

    Example:

    ```
    matchLabels:
      foo: bar
      baz: who
    ```

- **spec.match.namespaceSelector**
  object

  Specifies the Namespace selector to filter objects with.

  - **spec.match.namespaceSelector.excludeNames**
    array of strings

    Includes all namespaces except a particular set. Support glob pattern.

  - **spec.match.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces.

    You can get more info in the documentation.

    - **spec.match.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      The list of label expressions for namespaces.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      ```

- **spec.match.namespaceSelector.labelSelector.matchExpressions.key**
  string

  Required value

- **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**
  string

  Required value

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.namespaceSelector.labelSelector.matchExpressions.values**
  array of strings

- **spec.match.namespaceSelector.labelSelector.matchLabels**
  object

  The list of the labels that the namespace should have.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.match.namespaceSelector.matchNames**
  array of strings

  Includes only a particular set of namespaces. Supports glob pattern.

- **spec.policies**
  object

  Required value

  Policies that pods and containers must comply with.

  - **spec.policies.allowHostIPC**
    boolean

    Allows sharing the host’s IPC namespace with containers.

  - **spec.policies.allowHostNetwork**
    boolean

    Allows containers to use the host’s network.

  - **spec.policies.allowHostPID**
    boolean

    Allows sharing the host’s PID namespace with containers.

  - **spec.policies.allowPrivilegeEscalation**
    boolean

    Allows container processes to gain more privileges than its parent process.

    By default (if not specified) — `false`. This means that if the field is omitted or explicitly set to `false`, the container is denied any privilege escalation. If you want to allow elevated privileges, set this parameter to `true`.

  - **spec.policies.allowPrivileged**
    boolean

    Allows running containers in a privileged mode.

    By default (if not specified) — `false`. This means that if the field is omitted or explicitly set to `false`, the container is denied any privilege escalation. If you want to allow elevated privileges, set this parameter to `true`.

  - **spec.policies.allowRbacWildcards**
    boolean

    Allows using wildcards (`*`) in namespaced Role and RoleBinding resources.

    ClusterRole and ClusterRoleBinding are not validated by SecurityPolicy.

    By default (if not specified) — `true`. This means that if the field is omitted or explicitly set to `true`, using wildcards is allowed. If you want to deny wildcards, set this parameter to `false`.

    Default: `true`

  - **spec.policies.allowedAppArmor**
    array of strings

The list of AppArmor profiles the containers are permitted to use.

Example:

```
allowedAppArmor:
- runtime/default
- unconfined
```

- **Element of the array**
  string

  An AppArmor profile.

- **spec.policies.allowedCapabilities**
  array of strings

  The list of capabilities that the containers are permitted to use.

  To allow all capabilities, use `ALL`.

  Example:

  ```
  allowedCapabilities:
  - SETGID
  - SETUID
  - NET_BIND_SERVICE
  ```

  - **Element of the array**
    string

    A linux capability.

    Allowed values: `ALL`, `SETPCAP`, `SYS MODULE`, `SYS RAWIO`, `SYS PACCT`, `SYS ADMIN`, `SYS NICE`, `SYS RESOURCE`, `SYS TIME`, `SYS TTY CONFIG`, `MKNOD`, `AUDIT WRITE`, `AUDIT CONTROL`, `MAC OVERRIDE`, `MAC ADMIN`, `NET ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC_OVERRIDE`, `FOWNER`, `DAC_READ_SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX IMMUTABLE`, `NET BIND SERVICE`, `NET BROADCAST`, `IPC LOCK`, `IPC OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.allowedClusterRoles**
  array of strings

  A list of allowed cluster roles to bind to users.

- **spec.policies.allowedFlexVolumes**
  array of objects

  The list of Flex Volume drivers the containers are permitted to use.

  - **spec.policies.allowedFlexVolumes.driver**
    string

    A driver name.

- **spec.policies.allowedHostPaths**
  array of objects

  The list of allowed path prefixes for `hostPath` mounts. An empty list (`[]`) forbids `hostPath` for all paths. If the field is omitted, no restrictions apply.

  Example:

  ```
  allowedHostPaths:
  - pathPrefix: "/dev"
    readOnly: true
  ```

  - **spec.policies.allowedHostPaths.pathPrefix**
    string

    Required value

    The path prefix to match against the host volume.

    It does not support the `*` mask. Trailing slashes are trimmed when validating the path prefix with a host path.

    For example, the `/foo` prefix allows `/foo`, `/foo/` and `/foo/bar` path, but doesn’t allow `/food` or `/etc/foo` path.

  - **spec.policies.allowedHostPaths.readOnly**
    boolean

When set to true, allows host volumes to be matched against the pathPrefix only if all the volume mounts are read-only.

Default: `false`

- **spec.policies.allowedHostPorts**
  array of objects

  The list of `hostPort` ranges allowed by the rule. For pods with `hostNetwork: true`, an additional check is performed on the ports specified in `containerPort`.

  - **spec.policies.allowedHostPorts.max**
    integer

    Max value for the `hostPort`.

  - **spec.policies.allowedHostPorts.min**
    integer

    Min value for the `hostPort`.

- **spec.policies.allowedProcMount**
  string

  Allows `/proc` mount type for containers.

  Allowed values: `Default`, `Unmasked`

  Example:

  ```
  allowedProcMount: Unmasked.
  ```

- **spec.policies.allowedServiceTypes**
  array of strings

  The list of allowed service types.

  Example:

  ```
  allowedServiceTypes:
  - LoadBalancer
  - ClusterIP
  ```

  - **Element of the array**
    string

    Allowed values: `ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`

- **spec.policies.allowedUnsafeSysctls**
  array of strings

  The list of explicitly allowed unsafe sysctls.

  To allow all unsafe sysctls, use `*`.

  Example:

  ```
  allowedUnsafeSysctls:
  - kernel.msg*
  - net.core.somaxconn
  ```

- **spec.policies.allowedVolumes**
  array of strings

  The set of the permitted volume plugins.

  Example:

  ```
  allowedVolumes:
  - hostPath
  - persistentVolumeClaim
  ```

  - **Element of the array**
    string

    Allowed values: `*`, `none`, `awsElasticBlockStore`, `azureDisk`, `azureFile`, `cephFS`, `cinder`, `configMap`, `csi`, `downwardAPI`, `emptyDir`, `ephemeral`, `fc`, `flexVolume`, `flocker`, `gcePersistentDisk`, `gitRepo`, `glusterfs`, `hostPath`, `iscsi`, `nfs`, `persistentVolumeClaim`, `photonPersistentDisk`, `portworxVolume`, `projected`, `quobyte`, `rbd`, `scaleIO`, `secret`, `storageos`, `vsphereVolume`

- **spec.policies.automountServiceAccountToken**
  boolean

  Allows pods to run with `automountServiceAccountToken` enabled.

  When set to `false`, a pod is denied if it either sets `spec.automountServiceAccountToken: true` or mounts `/var/run/secrets/kubernetes.io/serviceaccount` in any container. The mount is checked regardless of the flag value, so explicitly projecting a service account token at that path is denied as well.

- **spec.policies.blockWildcardDomains**
  boolean

  Block creation of Ingress objects with `*` in DNS domains.

- **spec.policies.forbiddenSysctls**
  array of strings

  The list of forbidden sysctls.

  Takes precedence over allowed unsafe sysctls (allowedUnsafeSysctls).

  Example:

  ```
  forbiddenSysctls:
  - kernel.msg*
  - net.core.somaxconn
  ```

- **spec.policies.fsGroup**
  object

  Specifies which `fsGroup` values the security context is permitted to use.

  - **spec.policies.fsGroup.ranges**
    array of objects

    The list of `fsGroup` ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.fsGroup.ranges.max**
      integer

      Max ID value.

    - **spec.policies.fsGroup.ranges.min**
      integer

      Min ID value.

  - **spec.policies.fsGroup.rule**
    string

    Required value

    Specifies the strategy of the `fsGroup` selection.

    Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.readOnlyRootFilesystem**
  boolean

  If set to true, only the pods with the read-only root filesystem across all containers will be permitted to run. See the Kubernetes documentation for more details.

- **spec.policies.requiredDropCapabilities**
  array of strings

  The list of capabilities that have to be dropped from the containers.

  To exclude all capabilities, use `ALL`'.

  Example:

  ```
  requiredDropCapabilities:
  - SETGID
  - SETUID
  - NET_BIND_SERVICE
  ```

  - **Element of the array**
    string

    A linux capability to drop from the containers' specs.

    Allowed values: `ALL`, `SETPCAP`, `SYS_MODULE`, `SYS_RAWIO`, `SYS_PACCT`, `SYS_ADMIN`,

`SYS_NICE`, `SYS_RESOURCE`, `SYS_TIME`, `SYS_TTY_CONFIG`, `MKNOD`, `AUDIT_WRITE`, `AUDIT_CONTROL`, `MAC_OVERRIDE`, `MAC_ADMIN`, `NET_ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC_OVERRIDE`, `FOWNER`, `DAC_READ_SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX_IMMUTABLE`, `NET_BIND_SERVICE`, `NET_BROADCAST`, `IPC_LOCK`, `IPC_OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.runAsGroup**
  object

  Specifies which `runAsGroup` values the security context is permitted to use.

  - **spec.policies.runAsGroup.ranges**
    array of objects

    The list of group ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.runAsGroup.ranges.max**
      integer

      Max ID value.

    - **spec.policies.runAsGroup.ranges.min**
      integer

      Min ID value.

  - **spec.policies.runAsGroup.rule**
    string

    Required value

    Specifies the strategy of the group ID selection.

    Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.runAsUser**
  object

  Specifies which `runAsUser` values the security context is permitted to use.

  - **spec.policies.runAsUser.ranges**
    array of objects

    The list of user ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.runAsUser.ranges.max**
      integer

      Max ID value.

    - **spec.policies.runAsUser.ranges.min**
      integer

      Min ID value.

  - **spec.policies.runAsUser.rule**
    string

    Required value

    Specifies the strategy of the user ID selection.

    Allowed values: `MustRunAs`, `MustRunAsNonRoot`, `RunAsAny`

- **spec.policies.seLinux**
  array of objects

  Specifies which SELinux labels the security context is permitted to use.

  The `seLinuxOptions` of a pod or a container are allowed if they match at least one entry. A field left unset or empty in `seLinuxOptions` always matches. A field set in `seLinuxOptions` must equal the same field of the entry, so an entry that omits `user`, `role` or `type` forbids setting it. The `level` field is the exception: an entry that omits it allows any level.

  - **spec.policies.seLinux.level**
    string

    A SELinux level label that applies to the container.

  - **spec.policies.seLinux.role**
    string

    A SELinux role label that applies to the container.

  - **spec.policies.seLinux.type**
    string

A SELinux type label that applies to the container.

- **spec.policies.seLinux.user**
  string

  A SELinux user label that applies to the container.

- **spec.policies.seccompProfiles**
  object

  Specifies the list of allowed profiles that can be set for the Pod or container's seccomp annotations.

  - **spec.policies.seccompProfiles.allowedLocalhostFiles**
    array of strings

    Defines the local seccomp profiles (in JSON format) that can be used if `Localhost` is set in the `allowedProfiles` parameter.

    An empty list prohibits the use of any local profiles.

  - **spec.policies.seccompProfiles.allowedProfiles**
    array of strings

    The list of allowed profile values for seccomp on Pods/containers.

    Both formats are supported:

    - Via annotations: `runtime/default`, `docker/default`, `unconfined`, `localhost/some-profile.json`. `localhost/*` allows any local profile.
    - Via securityContext: `RuntimeDefault`, `Unconfined`, `Localhost`. For `Localhost`, specify the allowed profiles using the `allowedLocalhostFiles` parameter.

    Profile types:

    - `Unconfined` — no restrictions (not recommended for security reasons).
    - `RuntimeDefault` — the default profile provided by the container runtime (e.g., Docker, CRI-O).
    - `Localhost` — a custom profile defined on the host (flexible and tailored to the application).

    Using `*` allows all profiles. It's not necessary to specify both formats — they are automatically mapped to each other.

- **spec.policies.supplementalGroups**
  object

  Specifies what supplemental groups are allowed to be used by the security context.

  - **spec.policies.supplementalGroups.ranges**
    array of objects

    The list of supplemental group ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.supplementalGroups.ranges.max**
      integer

      Max ID value.

    - **spec.policies.supplementalGroups.ranges.min**
      integer

      Min ID value.

  - **spec.policies.supplementalGroups.rule**
    string

    Required value

    Specifies the strategy of the supplemental group ID selection.

    Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.verifyImageSignatures**
  array of objects

  Available in editions: SE+, Ultimate/EE

  List of policies to verify container images signatures.

  Container images must be signed using Cosign.

  Example:

```
verifyImageSignatures:
- reference: docker.io/myrepo/*
  publicKeys:
  - |-
    -----BEGIN PUBLIC KEY-----
    .....
    -----END PUBLIC KEY-----
- reference: company.registry.com/*
  dockerCfg: "<Base64_dockerCfg>"
  publicKeys:
  - |-
    -----BEGIN PUBLIC KEY-----
    .....
    -----END PUBLIC KEY-----
```

- **spec.policies.verifyImageSignatures.ca**
  string

  A custom certificate authority to use when connecting to the container image repository.

- **spec.policies.verifyImageSignatures.dockerCfg**
  string

  A string in Base64 with authentication data for the container image repository.

  If the container images are available anonymously, it is not specified.

- **spec.policies.verifyImageSignatures.publicKeys**
  array of strings

  Required value

  The list of Cosign compliant public keys.

- **spec.policies.verifyImageSignatures.reference**
  string

  Required value

  Absolute address or template for container images.

  If it contains the `*` symbol, it is considered a template for container image addresses. The `*` symbol can only appear once and only at the end. For example, for the value `company.registry.com/*`, any container images from the repository `company.registry.com` will be checked with the specified keys and access parameters.

  If it does not contain the `*` symbol, it is considered an absolute address for a container image. For example, for the value `company.registry.com/nginx`, only the nginx image from the repository `company.registry.com` will be checked with the specified keys and access parameters.

  Pattern: `^[a-z0-9\.\-:@\/]*\*?$`

  Examples:

  ```
  reference: docker.io/myuser/*
  ```

  ```
  reference: "*"
  ```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources (by Gatekeeper)

The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

Mutation Custom Resources

> **i** The `reinvocationPolicy: IfNeeded` is used in MutatingWebhookConfiguration. More details in the Kubernetes documentation.

Reference

Provide a configurable set of policies for modifying Kubernetes resources at the time they are deployed.

AssignMetadata

Reference

Allows you to modify the `Metadata` section of a resource. At the moment, Gatekeeper only allows **adding** `labels` and `annotations` objects. Modification of existing objects is not provided.

> **i** Using `*` in `spec.match.kinds` is not allowed. If `*` is specified, the mutation will not be applied. Instead, you must explicitly list the target resources (`kinds`) along with their corresponding `apiGroups`.

Example 1. Adding the `owner` label with the value `admin` to all namespaces:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: demo-annotation-owner
spec:
  match:
    scope: Namespaced
  location: "metadata.labels.owner"
  parameters:
    assign:
      value: "admin"
```

Example 2. Adding a label in a specific namespace and only to selected resources:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: set-labels-<your_namespace>
spec:
  match:
    scope: Namespaced
    namespaces: ["<your_namespace>"]
    kinds:
    - apiGroups: [""]
      kinds: ["Pod"] # The use of "*" is not allowed.
  location: "metadata.labels.<your_label_name>"
  parameters:
    assign:
      value: <your_label_value>
```

Assign

Allows you to modify fields outside the `Metadata` section.

An example of setting `imagePullPolicy` for all containers to `Always` in all namespaces except the `system` namespace:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: Assign
metadata:
  name: demo-image-pull-policy
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    scope: Namespaced
    kinds:
    - apiGroups: ["*"]
      kinds: ["Pod"]
    excludedNamespaces: ["system"]
  location: "spec.containers[name:*].imagePullPolicy"
  parameters:
    assign:
      value: Always
```

ModifySet

Reference

Allows you to add and remove items from a list, such as arguments for running a container. New values are added to the end of the list.

An example of removing the `--alsologtostderr` argument from all containers in a pod:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: ModifySet
metadata:
  name: remove-err-logging
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  location: "spec.containers[name: *].args"
  parameters:
    operation: prune
    values:
      fromList:
        - --alsologtostderr
```

AssignImage

Reference

Allows you to make changes to the `image` parameter of a resource.

An example of changing the `image` parameter to the value `my.registry.io/repo/app@sha256:abcde67890123456789abc345678901a`:

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: AssignImage
metadata:
  name: assign-container-image
spec:
  applyTo:
  - groups: [ "" ]
    kinds: [ "Pod" ]
    versions: [ "v1" ]
  location: "spec.containers[name:*].image"
  parameters:
    assignDomain: "my.registry.io"
    assignPath: "repo/app"
    assignTag: "@sha256:abcde67890123456789abc345678901a"
  match:
    source: "All"
    scope: Namespaced
    kinds:
    - apiGroups: [ "*" ]
      kinds: [ "Pod" ]
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

The module lifecycle stage: General Availability

How do I configure alternative security policy management solutions?

For DP to work correctly, extended privileges are required to run and operate system component payloads. If you are using some alternative security policy management solution (e.g., Kyverno) instead of the admission-policy-engine module, you have to configure exceptions for the following namespaces:

- `kube-system`;
- all namespaces with the `d8-*` prefix (e.g., `d8-system`).

How do I configure policy selectors?

In `OperationPolicy` and `SecurityPolicy`, the `spec.match` field determines which specific objects (pods) in the cluster the policy will apply to. It must be present in the configuration. Filtering is performed by combining two main criteria: a pod selector (`labelSelector`) and a namespace selector (`namespaceSelector`).

If both selectors are specified, the policy is applied only to those pods that simultaneously:

- satisfy the pod selection conditions.
- are located in namespaces that passed the filtering.

If either selector is omitted, the corresponding check is not performed (all pods or namespaces will be used).

- `spec.match.labelSelector` – pod selection

The `labelSelector` is used to set criteria for selecting pods by their labels. Two mutually exclusive methods are supported:

- `matchLabels` – a simple check for an exact label match (key-value). The pod must have all the specified labels.
- `matchExpressions` – flexible expressions with operators. Each expression is defined by an object with the following fields:
  - `key` (string, required) – the label name.
  - `operator` (string, required) – one of the values: `In`, `NotIn`, `Exists`, `DoesNotExist`.
  - `values` (array of strings) – a list of values for the `In` / `NotIn` operators; not specified for `Exists` / `DoesNotExist`.

All elements in the `matchExpressions` list are combined with a logical AND – the pod must satisfy every expression.

Examples:

```
spec:
  match:
    labelSelector:
      matchLabels:
        app: nginx
        role: frontend
```

```
spec:
  match:
    labelSelector:
      matchExpressions:
        - key: tier
          operator: In
          values:
            - production
            - staging
        - key: monitoring
          operator: Exists
```

- `spec.match.namespaceSelector` – namespace selection

Allows limiting the namespaces in which the policy is active. You can use a combination of three filters:

- `matchNames` – an explicit list of allowed namespaces. If specified, the policy only applies to the listed namespaces.
- `excludeNames` – a list of excluded namespaces. The policy will apply to all namespaces except those specified.
- `labelSelector` – a selector based on the labels of the Namespace object itself.

If multiple fields are set, they are combined with logical AND:

1. Start with the set from `matchNames` (or all namespaces if `matchNames` is omitted).
2. Apply `labelSelector` (if set).
3. Subtract `excludeNames`.

Formula:

`result = (base_from_matchNames ∩ selected_by_labelSelector) \ excludeNames`

It is recommended not to mix `matchNames`, `excludeNames`, and `labelSelector` without a clear need.

When to use `labelSelector`

Use `labelSelector` when the policy should automatically apply to a group of namespaces by attribute rather than by fixed names.

For example:

- “all namespaces with label `env=prod`”;
- “all `team=backend` namespaces” (with the corresponding label);
- “all namespaces with `security.deckhouse.io/pod-policy=restricted`”.

`labelSelector` is especially useful when namespaces are created/deleted dynamically: it is enough to automatically set a label on the namespace when it is created, and the policy will apply without editing the policy.

`labelSelector` is not required if you have a small static list of namespaces — in that case, using `matchNames` is simpler and easier to read.

Can `matchNames` and `labelSelector` be used together?

Yes, technically they are not mutually exclusive: you can specify both, and then their intersection applies.

But in practice this often hurts readability and complicates maintenance. Therefore, it is recommended to choose one primary selection method:

- either `matchNames + excludeNames`;
- or `labelSelector + excludeNames`.

This makes it easier to understand why a particular namespace is included in or excluded from the policy.

Typical scenarios

1. Static environments list → `matchNames + excludeNames`.
2. Dynamic environments/teams → `labelSelector + excludeNames`.

3. `matchNames + labelSelector` combination — only when you really need the intersection of two independent conditions.

Examples:

Static namespace list (`matchNames + excludeNames`)

```
spec:
  match:
    namespaceSelector:
      matchNames:
        - production
        - staging
      excludeNames:
        - staging
```

Result: the policy applies only in the `production` namespace.

Dynamic label-based selection (`labelSelector + excludeNames`)

```
spec:
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          team: backend
          environment: production
      excludeNames:
        - backend-sandbox
```

Result: all namespaces with labels `team=backend` and `environment=production`, except `backend-sandbox`.

Flexible filtering with expressions (`labelSelector.matchExpressions`)

```
spec:
  match:
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: compliance
            operator: In
            values:
              - pci
              - sox
          - key: lifecycle
            operator: NotIn
            values:
              - deprecated
```

Result: only namespaces with labels `compliance=pci` or `compliance=sox` and without label `lifecycle=deprecated`.

`matchNames` and `labelSelector` combination (intersection, use carefully)

```
spec:
  match:
    namespaceSelector:
      matchNames:
        - production
        - staging
        - qa
      labelSelector:
        matchLabels:
          team: backend
```

Result: applies only to namespaces that simultaneously:

- are in the list `production|staging|qa`;
- have label `team=backend`.

For example, if `qa` does not have `team=backend`, it will not be matched.

How do I extend Pod Security Standards policies?

> **i** Pod Security Standards respond to the `security.deckhouse.io/pod-policy: restricted` or `security.deckhouse.io/pod-policy: baseline` label.

To extend the Pod Security Standards policy by adding your checks to existing checks, you need to:

- Create a constraint template for the check (`ConstraintTemplate`).
- Bind it to the `restricted` or `baseline` policy.

Example of the `ConstraintTemplate` for checking a repository URL of a container image:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }
```

Example of binding a check to the `restricted` policy:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"
```

The example demonstrates the configuration of checking the repository address in the `image` field for all Pods created in the namespace having the `security.deckhouse.io/pod-policy: restricted` label. A Pod will not be created if the address in the `image` field of the Pod does not start with `mycompany.registry.com`.

For more information about templates and the policy language, see the Gatekeeper documentation.

More examples of checks for policy extension are available in the Gatekeeper Library.

How to allow some Pod Security Standards policies without disabling whole list?

To apply only the required security policies without turning off the entire built-in set:

1. Add the `security.deckhouse.io/pod-policy: privileged` label to your namespace in order to disable built-in policies.
2. Create a SecurityPolicy that matches the baseline or restricted policy while also editing the list of `policies` elements as you see fit. Take the rules from the `d8-pod-security-baseline` or `d8-pod-security-restricted` object the module publishes the standards as, see Rules of the standards.
3. Add a label to your namespace that matches the `namespaceSelector` in the SecurityPolicy. In the examples below, the label is `security-policy.deckhouse.io/baseline-enabled: "true"` or `security-policy.deckhouse.io/restricted-enabled: "true"`.

> **i** The `allowedHostPaths` field defines the list of allowed path prefixes for `hostPath` mounts. An empty list (`[]`) forbids `hostPath` for all paths. If the field is omitted, no restrictions apply.

SecurityPolicy that matches baseline:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - AUDIT_WRITE
      - CHOWN
      - DAC_OVERRIDE
      - FOWNER
      - FSETID
      - KILL
      - MKNOD
      - NET_BIND_SERVICE
      - SETFCAP
      - SETGID
      - SETPCAP
      - SETUID
      - SYS_CHROOT
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
        - undefined
        - ''
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/baseline-enabled: "true"
```

SecurityPolicy that matches restricted:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: restricted
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
    requiredDropCapabilities:
      - ALL
    runAsUser:
      rule: MustRunAsNonRoot
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/restricted-enabled: "true"
```

What if there are multiple policies (operational or security) that are applied to the same object?

In that case the object's specification has to fulfill all the requirements imposed by the policies.

For example, consider the following two security policies:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: foo
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - MKNOD
---
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: bar
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    requiredDropCapabilities:
    - NET_BIND_SERVICE
```

Then, in order to fulfill the requirements of the above security policies, the following settings must be set in a container specification:

```
    securityContext:
      capabilities:
        drop:
          - MKNOD
          - NET_BIND_SERVICE
      readOnlyRootFilesystem: true
```

Why is an Assign or AssignMetadata mutation rejected or silently ignored?

The `Assign`, `AssignMetadata`, `ModifySet`, and `AssignImage` CRDs are imported from Gatekeeper without modification (see Mutation Custom Resources). However, the Gatekeeper mutation webhook enforces content restrictions that go beyond the CRD schema. The `admission-policy-engine` module enforces the checkable restrictions with built-in ValidatingAdmissionPolicies. As a result, `kubectl apply` rejects an invalid resource immediately instead of silently ignoring it or having Gatekeeper reject it later with an unclear parser error.

The module enforces the following restrictions:

- `deny-invalid-assign-location.deckhouse.io` — the `spec.location` field of `Assign` cannot target the `metadata` field (`metadata.name`, `.namespace`, `.labels`, `.annotations`). Use `AssignMetadata` to add labels or annotations.
- `deny-invalid-assignmetadata-location.deckhouse.io` — the `spec.location` field of `AssignMetadata` must be exactly `metadata.labels.<KEY>` or `metadata.annotations.<KEY>`. If `<KEY>` contains characters other than letters, digits, `_`, or `-` (for example, a domain-prefixed key such as `app.kubernetes.io/name`), wrap it in quotes: `metadata.annotations."app.kubernetes.io/name"`.
- `deny-invalid-assignmetadata-value.deckhouse.io` — the `spec.parameters.assign.value` field of `AssignMetadata` must be a string, because Kubernetes label and annotation values are strings.
- `deny-invalid-mutator-frommetadata-field.deckhouse.io` — the `spec.parameters.assign.fromMetadata.field` field (in `Assign` and `AssignMetadata`) is mandatory once `fromMetadata` is set, and accepts only `namespace` or `name`.
- `deny-invalid-assignmetadata-externaldata-datasource.deckhouse.io` — the `spec.parameters.assign.externalData.dataSource` field of `AssignMetadata` supports only `Username`. This field defaults to `ValueAtLocation` when omitted, and that default is rejected too — set `dataSource: Username` explicitly. Unlike `Assign`, `AssignMetadata` doesn't support `ValueAtLocation`, because a

metadata mutation has no source value at `location` to read.
- `deny-mutator-without-match-kinds.deckhouse.io` — the `spec.match.kinds` field is required and must not be empty in `Assign`, `AssignMetadata`, `ModifySet`, and `AssignImage` resources, and every entry in it must set non-empty `apiGroups` and `kinds` without a `*` kind. This restriction is specific to Deckhouse: the `d8-admission-policy-engine-config` MutatingWebhookConfiguration rules are generated only from the `match.kinds` field across all resources of these kinds, resolving `apiGroups`/`kinds` through the Kubernetes REST mapper. A resource with an entry that doesn't resolve to a concrete resource type never triggers a mutation, even though Gatekeeper itself would still treat it as valid.

How to apply a mutation only when a resource is created?

Use the `spec.applyTo[].operations` field of `Assign`, `ModifySet`, or `AssignImage` to choose the admission operations that trigger the mutation. The field accepts `CREATE`, `UPDATE`, and `*`. If the field is omitted or contains `*`, the mutation runs on both `CREATE` and `UPDATE`.

The following example sets `imagePullPolicy` only when a pod is created and leaves updates of the pod unchanged:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: Assign
metadata:
  name: always-pull-on-create
spec:
  applyTo:
    - groups: [""]
      kinds: ["Pod"]
      versions: ["v1"]
      operations: ["CREATE"]
  match:
    scope: Namespaced
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  location: "spec.containers[name:*].imagePullPolicy"
  parameters:
    assign:
      value: Always
```

The module builds the rules of the `d8-admission-policy-engine-config` MutatingWebhookConfiguration from these operations. A resource is sent to Gatekeeper only for the operations that at least one of its mutations requests. `AssignMetadata` has no `applyTo` field and always runs on `CREATE` and `UPDATE`.

Verification of image signatures

The module implements a function for verifying signatures of container images signed using Cosign. For more details on signing and verifying container images, see the DP documentation.

Signature verification caches the credentials of a private registry read from the `dockerCfg` field of a SecurityPolicy. After you change `dockerCfg`, the new credentials take effect once the cache entry expires. To set the cache lifetime, use the `imageSignatureVerification.registryCredentialsCacheTTLSeconds` parameter. The default is 300 seconds.

How to block deleting a node without a label

> **i** DELETE operations are handled by Gatekeeper by default.

You can create your own Gatekeeper policy to block Node deletion unless a special label is present. The example below uses `oldObject` to check labels on the Node being deleted:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Node deletion is blocked. Add label %q=%q to proceed.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"
```

How to deny kubectl exec and kubectl attach to specific Pods?

The `admission-policy-engine` module webhook routes `CONNECT` requests for `pods/exec` and `pods/attach` in namespaces other than `d8-*` and `kube-*` through Gatekeeper. This allows creating custom policies to allow or deny `kubectl exec` and `kubectl attach` operations.

The `deckhouse` module denies the following operations into pods of system components labeled `heritage: deckhouse` for all users, but the Deckhouse Platform system users:

- `kubectl exec`
- `kubectl attach`
- `kubectl cp`
- `kubectl port-forward`
- `kubectl debug`

The operations are denied regardless of whether the `admission-policy-engine` module is enabled in the

cluster. For details, refer to the FAQ of the `deckhouse` module.

Built-in policy for finalizers

To protect objects managed by DP controllers, the `admission-policy-engine` module includes a built-in ValidatingAdmissionPolicy `deny-deckhouse-finalizers.deckhouse.io` that forbids removing finalizers containing the `deckhouse.io` substring on any cluster objects.

This policy doesn't apply to the following users who are allowed to remove such finalizers:

- Kubernetes system controllers (`system:kube-controller-manager`, `system:kube-scheduler`, etc.);
- `system:sudouser`, `dhctl`, `observability`;
- service accounts from `d8-*` namespaces (`system:serviceaccount:d8-*`);
- service accounts from `kube-*` namespaces (`system:serviceaccount:kube-*`).

Custom policy example

You can create your own Gatekeeper policy to deny `kubectl exec` and `kubectl attach` operations in specific namespaces. The example below uses `input.review.operation` and `input.review.resource.resource` to check for `CONNECT` operations:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource is preferred, but fall back to subResource for older APIs
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource", input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
        }

        violation[{"msg": msg}] {
          is_connect
          is_exec_or_attach
```

```
        is_forbidden_namespace
        msg := sprintf("Exec/attach is forbidden in namespace %q", [input.review.namespace])
      }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
  name: deny-exec-in-namespaces
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: ["*"]
        kinds: ["*"]
    scope: Namespaced
  parameters:
    forbiddenNamespaces:
      - production
      - staging
```

Key data and checks available when validating `CONNECT` operations:

- Use `input.review.operation == "CONNECT"` to check for `CONNECT` operations.
- User information is available in `input.review.userInfo.username` and `input.review.userInfo.groups`.
- The namespace is available in `input.review.namespace`.

How do I restrict GPU resource usage in namespaces?

The `gpuResourceRestriction` policy in OperationPolicy denies the workloads that request GPU resources when the namespace has no label allowing GPU usage. The check runs on creation and update of a pod, including when an ephemeral container is added, and on creation and update of the controllers that create pods. Both `resources.requests` and `resources.limits` of every container, init container, and ephemeral container are inspected. A resource with a quantity of `0` is not treated as a GPU request.

To set up GPU resource restriction:

1. Create an OperationPolicy with the `gpuResourceRestriction` setting.

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: OperationPolicy
   metadata:
     name: gpu-restriction
   spec:
     enforcementAction: Deny
     match:
       namespaceSelector:
         labelSelector:
           matchLabels:
             operation-policy.deckhouse.io/enabled: "true"
     policies:
       gpuResourceRestriction:
         namespaceLabel:
           key: "gpu.deckhouse.io/enabled"
           value: "true"
         gpuResourcePatterns:
           - '^nvidia\.com/.*$'
           - '^amd\.com/gpu$'
   ```

   The `namespaceLabel` parameter defines the label key and value that must be present on the namespace. The `gpuResourcePatterns` parameter lists the regular expressions matched against GPU resource names. Patterns are not anchored implicitly, so use `^` and `$` to match the whole resource name.

2. Add the policy label to the namespaces where the policy must be enforced.

   ```
   d8 k label ns my-namespace operation-policy.deckhouse.io/enabled=true
   ```

3. Add the GPU label to the namespaces where GPU usage is allowed.

   ```
   d8 k label ns my-gpu-namespace gpu.deckhouse.io/enabled=true
   ```

After that, the pods that request GPU resources are allowed only in the namespaces with the `gpu.deckhouse.io/enabled: "true"` label.

⚠ The policy is applied to pods and to the pod-creating controllers, so a Deployment whose pod template requests GPU resources is denied on creation. To check this rule on Pods only, add `gpuResourceRestriction` to the `spec.controllerValidation.excludedPolicies` parameter of the policy.

The namespace label is read from the Gatekeeper cache. While a namespace is missing from that cache, for example when the namespace and the pod are applied together and the namespace has not been cached yet, the policy denies the pods that request GPU resources in that namespace.

The devices requested through Dynamic Resource Allocation in the `spec.resourceClaims` field are not extended resources, and the policy does not check them.

What to do if the validating webhook is unavailable?

While the `gatekeeper-controller-manager` Deployment has no available replicas, the API server rejects every request its webhook intercepts, and the cluster cannot create or delete workloads. The `D8AdmissionPolicyEngineWebhookUnavailable` alert reports this state. Read why the component is critical for the list of what stops working.

Confirm that the Deployment is the cause and collect the reason:

```
d8 k -n d8-admission-policy-engine get deploy gatekeeper-controller-manager
d8 k -n d8-admission-policy-engine get pods -l app=gatekeeper,control-plane=controller-manager -o wide
d8 k -n d8-admission-policy-engine describe pods -l app=gatekeeper,control-plane=controller-manager
d8 k -n d8-admission-policy-engine logs deploy/gatekeeper-controller-manager -c manager --all-pods=true --
tail=200
```

The pods of this Deployment are excluded from validation by their `gatekeeper.sh/operation: webhook` label, so the ReplicaSet creates a replacement pod even while the webhook is down. To get a fresh pod, delete the current one:

```
d8 k -n d8-admission-policy-engine delete pod -l app=gatekeeper,control-plane=controller-manager
```

Restarting the Deployment with `d8 k rollout restart` does not work here. The same label is guarded by the `deny-gatekeeper-webhook-operation-label-controllers` ValidatingAdmissionPolicy, which rejects a change to a pod template carrying it unless the request comes from a service account of a `d8-*` or `kube-*` namespace, or from `system:sudouser`.

Look for the cause first. Disabling the module unblocks the cluster, but it also removes every policy the cluster relies on, so treat it as a last resort rather than as the first step:

```
d8 platform module disable admission-policy-engine
```

Deckhouse Platform keeps the objects of the module removed while it is disabled, so the cluster stays unblocked for as long as the module is off. Return the module as soon as the cause is fixed, since no policy is enforced while it is disabled:

```
d8 platform module enable admission-policy-engine
```

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The cert-manager module

ℹ Included in extensions: **Cluster Union**

ℹ The module lifecycle stage: General Availability

This module installs the reliable and highly available cert-manager release 1.21.2.

The installation process automatically takes into account cluster aspects:

- the component (webhook) that the `kube-apiserver` is accessing is installed on master nodes;
- if the webhook is unavailable, the `apiservice` is temporary deleted so that the unavailability of *cert-manager* does not block regular cluster operation.

The module itself is updated automatically (including the migration of cert-manager resources).

Features of the cert-manager module (with the changes made)

The module has all the features of the original cert-manager, including:

- Provisioning certificates of all the supported CA such as *Let's Encrypt*, *HashiCorp Vault*, *Venafi*;
- Issuing self-signed certificates;
- Keeping certificates up-to-date, reissuing them automatically, etc.

Changes to the original cert-manager were made so that the `cm-acme-http-solver` Pods could run on master and dedicated nodes.

Restricting ClusterIssuers for projects

When the `multitenancy-manager` module is enabled, `cert-manager` registers ClusterIssuer as a cluster-wide resource whose use from project namespaces can be restricted. The registration is the `clusterissuers` GrantableClusterResourceDefinition. It applies to the following references:

- the `.spec.issuerRef.name` field of a Certificate if `.spec.issuerRef.kind` is set to ClusterIssuer;
- the `cert-manager.io/cluster-issuer` annotation of an Ingress.

By default, every ClusterIssuer is available to every project. To limit the list or set the ClusterIssuer used by default, create a ClusterResourceGrantPolicy with `resourceName: clusterissuers`.

Monitoring

The module can expose metrics in the Prometheus format, allowing you to monitor:

- certificate validity;
- correctness of the certificate reissue.

Module roles

The module has several well-thought-out roles for managing resources:

- `User` – has read-only access to Certificate & Issuers resources in the permitted namespaces and to the global clusterIssues;
- `Editor` – manages Certificate and Issuer resources in the permitted namespaces;
- `ClusterEditor` – manages Certificate & Issuer resources in all namespaces;
- `SuperAdmin` – manages internal service objects.

Third-party components

List of third-party software used in the `cert-manager` module:

- cert-manager 1.21.2

  License: Apache License 2.0

  A Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

- cert-manager-webhook-yandex 1a7dd500a97e9206fa5cfebb102decd3889e3544

  License: Apache License 2.0

  ACME DNS-01 webhook for Yandex Cloud DNS.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Included in extensions: **Cluster Union**

> **i** The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

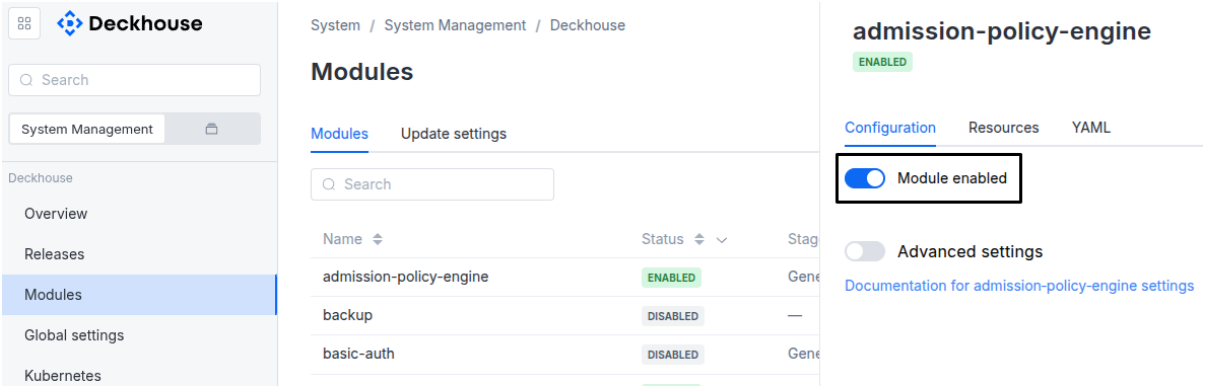
The module has 6 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cert-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cert-manager
```

- **Using ModuleConfig `cert-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cert-manager` (create it if necessary);

  Example of a manifest to enable module `cert-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  enabled: true
```

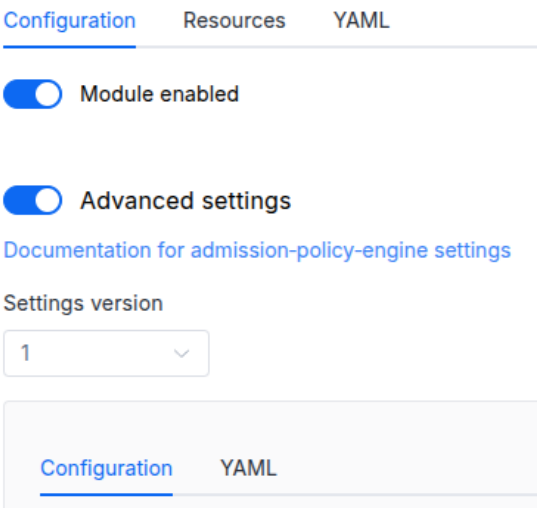
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cert-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cert-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cert-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cert-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cert-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cert-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cert-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cert-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.certificateRequestMaximumBackoffDuration**
    string

    Maximum delay between attempts to issue a certificate after a failed CertificateRequest.

    The delay starts at one hour and doubles after each consecutive failure, but does not exceed this value. A lower value makes cert-manager retry sooner once the issuer is available again, for example after maintenance of the CA. Specify the value in the `<HOURS>h` or `<HOURS>h<MINUTES>m` format, for example `6h` or `1h30m`. The number of hours is from 1 to 9999, the number of minutes is from 0 to 59. Other notations of the same duration, such as `90m` or `1h30m0s`, are not accepted.

    If the parameter is not set, the cert-manager default of 32 hours applies.

    Pattern: `^[1-9][0-9]{0,3}h([0-5]?[0-9]m)?$`

    Examples:

```
certificateRequestMaximumBackoffDuration: 6h
```

```
certificateRequestMaximumBackoffDuration: 1h30m
```

- **settings.cleanupOrphanSecrets**
  boolean

  Delete a secret with a certificate automatically if the corresponding Certificate resource was deleted from the cluster.

  Default: `false`

  Examples:

  ```
  cleanupOrphanSecrets: true
  ```

  ```
  cleanupOrphanSecrets: false
  ```

- **settings.cloudDNSServiceAccount**
  string

  The Service Account for Google Cloud for the same project that has the DNS Administrator role.

  Example:

  ```
  cloudDNSServiceAccount:
  eyJzYSI6ICJhYmNkZWZnaEBzZXJ2aWNlYWNjb3VudC5jb20iLCAicHJvamVjdF9pZCI6ImFhYWFhIn0=
  ```

- **settings.cloudflareAPIToken**
  string

  API Tokens allow application-scoped keys bound to specific DNS zones.

  API Tokens are recommended for higher security, since they have more restrictive permissions and are more easily revocable.

  It allows you to verify that domains specified in the Certificate resource are managed by `cert-manager` and kept by the Cloudflare DNS provider. Verification is performed by adding special TXT records for the ACME DNS01 Challenge Provider domain.

  Example:

  ```
  cloudflareAPIToken: token
  ```

- **settings.cloudflareEmail**
  string

  The email used for accessing the Cloudflare platform.

  Example:

  ```
  cloudflareEmail: example@example.com
  ```

- **settings.cloudflareGlobalAPIKey**
  string

  The Cloudflare Global API key for managing DNS records

  It allows you to verify that domains specified in the Certificate resource are managed by `cert-manager` and kept by the Cloudflare DNS provider.

  Verification is performed by adding special TXT records for the ACME DNS01 Challenge Provider domain.

  Example:

  ```
  cloudflareGlobalAPIKey: key
  ```

- **settings.digitalOceanCredentials**
  string

  The Access Token for the Digital Ocean API (you can create it in the `API` section).

  Example:

  ```
  digitalOceanCredentials: creds
  ```

- **settings.disableLetsencrypt**
  boolean

  Disable `letsencrypt` and `letsencrypt-staging` ClusterIssuer objects (if set to `true`).

  Examples:

  ```
  disableLetsencrypt: true
  ```

  ```
  disableLetsencrypt: false
  ```

- **settings.email**
  string

  The email used for sending notifications by LetsEncrypt.

  Example:

  ```
  email: example@example.com
  ```

- **settings.enableACMERenewalInfo**
  boolean

  Enable ACME Renewal Information (ARI, RFC 9773) by turning on the `ACMEUseARI` feature gate.

  With ARI, cert-manager asks the ACME server, for example Let’s Encrypt, for the renewal window of every certificate it issued. If the server returns a window, cert-manager renews the certificate at a random moment inside it and ignores `renewBefore` and `renewBeforePercentage` of the Certificate resource. The renewal can therefore happen later than `renewBefore` requires. `spec.renewal` of the Certificate still applies on top of the window. If the server does not support ARI or the request fails, `renewBefore` applies as usual.

  The parameter affects every ACME Issuer and ClusterIssuer in the cluster, not only the ones of the module. The feature is in the alpha stage in cert-manager.

  Default: `false`

  Examples:

  ```
  enableACMERenewalInfo: true
  ```

  ```
  enableACMERenewalInfo: false
  ```

- **settings.enableCAInjector**
  boolean

  Manually enable CAInjector. By default, CAInjector is enabled automatically when resources with CA injection annotations are detected (`cert-manager.io/inject-ca-from`, `cert-manager.io/inject-ca-from-secret`, `cert-manager.io/inject-apiserver-ca`).

  Default: `false`

  Examples:

  ```
  enableCAInjector: true
  ```

  ```
  enableCAInjector: false
  ```

- **settings.enableListenerSet**
  boolean

  Enable ListenerSet support in cert-manager by turning on the `ListenerSets` feature gate.

  Additionally, when Gateway API integration is enabled, this flag also enables ListenerSet integration for Gateway API if the ListenerSet CRD is installed in the cluster and serves the `v1` version.

  Gateway API integration is enabled automatically when the Gateway and HTTPRoute CRDs are installed in the cluster and serve the `v1` version.

  Default: `false`

  Examples:

  ```
  enableListenerSet: true
  ```

```
enableListenerSet: false
```

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

  Examples:

  ```
  highAvailability: true
  ```

  ```
  highAvailability: false
  ```

- **settings.http01WaitInsteadOfSelfCheck**
  string

  Time to wait instead of the self-check before asking the ACME server to validate an HTTP-01 challenge.

  Applies to the `letsencrypt` and `letsencrypt-staging` ClusterIssuers, including their Gateway API variants. Before asking the ACME server, cert-manager checks that the challenge is reachable from inside the cluster. This self-check fails when the cluster cannot reach its own public address, for example behind NAT without hairpinning (when you cannot access your external IP address from inside the cluster) or with split-horizon DNS (when the same domain resolves to different IP addresses depending on where the DNS request originated: from inside the cluster or from outside), although the ACME server could validate the challenge. If the parameter is set, cert-manager skips the self-check and waits for the specified time instead. The value `0` asks the ACME server to validate right away and relies on its own retries.

  By default, the self-check is performed.

  Pattern: `^(0|[0-9]{1,4}h([0-9]{1,4}m)?([0-9]{1,4}s)?|[0-9]{1,4}m([0-9]{1,4}s)?|[0-9]{1,4}s)$`

  Examples:

  ```
  http01WaitInsteadOfSelfCheck: 30s
  ```

  ```
  http01WaitInsteadOfSelfCheck: 2m
  ```

- **settings.ingressClassHttp01**
  string

  The name of the `ingressClass` used to confirm ownership of domain using the ACME HTTP-01 challenges method. If the parameter is omitted, the default `ingressClass` is used.

  Example:

  ```
  ingressClassHttp01: nginx
  ```

- **settings.letsencrypt**
  object

  Settings for the `letsencrypt` ClusterIssuer.

  - **settings.letsencrypt.resources**
    object

    Resource requests and limits (CPU and memory) for the workload using the `letsencrypt` ClusterIssuer.

    Example:

    ```
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    ```

    - **settings.letsencrypt.resources.limits**
      object
      - **settings.letsencrypt.resources.limits.cpu**

string

Required value

The same as in the pods' `spec.containers.resources.limits.cpu` parameter in Kubernetes.

Default: `50m`

Pattern: `^[0-9]+m?$`

- **settings.letsencrypt.resources.limits.memory**
  string or number

  Required value

  The same as in the pods' `spec.containers.resources.limits.memory` parameter in Kubernetes.

  Default: `64Mi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencrypt.resources.requests**
  object

  Resource requests and limits (CPU and memory) for pods that create the `letsencrypt` ClusterIssuer for http01 challenges

  - **settings.letsencrypt.resources.requests.cpu**
    string

    Required value

    The same as in the pods' `spec.containers.resources.requests.cpu` parameter in Kubernetes.

    Default: `10m`

    Pattern: `^[0-9]+m?$`

  - **settings.letsencrypt.resources.requests.memory**
    string or number

    Required value

    The same as in the pods' `spec.containers.resources.requests.memory` parameter in Kubernetes.

    Default: `32Mi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencryptStaging**
  object

  Settings for the `letsencrypt-staging` ClusterIssuer.

  - **settings.letsencryptStaging.resources**
    object

    Resource requests and limits (CPU and memory) for pods that create the `letsencrypt-staging` ClusterIssuer for http01 challenges.

    Example:

    ```
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    ```

    - **settings.letsencryptStaging.resources.limits**
      object
      - **settings.letsencryptStaging.resources.limits.cpu**
        string

        Required value

        The same as in the pods' `spec.containers.resources.limits.cpu` parameter in Kubernetes.

Default: `50m`

Pattern: `^[0-9]+m?$`

- **settings.letsencryptStaging.resources.limits.memory**
  string or number

  Required value

  The same as in the pods' `spec.containers.resources.limits.memory` parameter in Kubernetes.

  Default: `64Mi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencryptStaging.resources.requests**
  object
  - **settings.letsencryptStaging.resources.requests.cpu**
    string

    Required value

    The same as in the pods' `spec.containers.resources.requests.cpu` parameter in Kubernetes.

    Default: `10m`

    Pattern: `^[0-9]+m?$`

  - **settings.letsencryptStaging.resources.requests.memory**
    string or number

    Required value

    The same as in the pods' `spec.containers.resources.requests.memory` parameter in Kubernetes.

    Default: `32Mi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.maxConcurrentChallenges**
  integer

  The maximum number of challenges that can be scheduled as 'processing' at once. (default 60)

  Allowed values: `0 <= X`

  Example:

  ```
  maxConcurrentChallenges: 25
  ```

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    has-gpu: 'true'
  ```

- **settings.recursiveSettings**
  object

  Parameters for using recursive DNS servers, which are used to check the existence of a DNS record, before initiating the domain ownership verification process using the ACME DNS-01 method.

  Example:

  ```
  recursiveSettings:
    nameservers:
    - 8.8.8.8:53
    - https://1.1.1.1
    useOnlyRecursive: true
  ```

  - **settings.recursiveSettings.nameservers**
    array of strings

List of recursive DNS servers (IP:port or HTTPS URLs).

- **Element of the array**
  string

  Pattern: `^(?:[0-9]{1,3}(?:\.[0-9]{1,3}){3}:\d{1,5}|https?://[^\s]+)$`

- **settings.recursiveSettings.useOnlyRecursive**
  boolean

  When set to true, only the specified recursive DNS servers will be used to check for the existence of a record in the DNS before starting the domain ownership verification process.

  Default: `false`

- **settings.route53AccessKeyID**
  string

  The Access Key ID of the user with the attached Amazon Route53 IAM Policy for managing domain records.

  Example:

  ```
  route53AccessKeyID: key_id
  ```

- **settings.route53SecretAccessKey**
  string

  The Secret Access Key of the user with privileges to manage domain records.

  Example:

  ```
  route53SecretAccessKey: secret
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: dedicated.deckhouse.io
    operator: Equal
    value: cert-manager
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

- **settings.yandexFolderID**
  string

  ID of the Yandex Cloud folder that contains the public DNS zone used for the ACME DNS-01 challenge.

  Must be set together with `yandexServiceAccountJSON`.

  Example:

  ```
  yandexFolderID: b1gabcdefghijklmnopq
  ```

- **settings.yandexServiceAccountJSON**
  string

  Base64-encoded authorized key JSON for a Yandex Cloud service account with the `dns.editor` role (or equivalent) on the folder with the DNS zone.

  Must be set together with `yandexFolderID`. Details are available in Issuing a DNS wildcard certificate using Yandex Cloud DNS.

  Example:

```
yandexServiceAccountJSON: eyJpZCI6ImFqZSIsInNlcnZpY2VfYWNjb3VudF9pZCI6ImFqZSJ9
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

The module uses the standard cert-manager Custom Resource called Certificate to request certificates.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> The module lifecycle stage: General Availability

An example of provisioning a certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com                     # the name of the certificate; you can use it to view
the cert's status
  namespace: default
spec:
  secretName: example-com-tls             # the name of the secret to store a private key and
a certificate
  issuerRef:
    kind: ClusterIssuer                  # the link to the certificate "issuer", see more
below
    name: letsencrypt
  commonName: example.com                 # the main certificate domain
  dnsNames:                              # additional domains (At least one DNS Name or IP
address is required)
  - www.example.com
  - admin.example.com
```

Here:

- a separate Ingress resource is created for the duration of the challenge (thus, authentication and whitelist of the primary Ingress will not interfere with the process),
- you can issue a single certificate for several Ingress resources (the deletion of the resource based on the `tls-acme` annotation won't affect it in any way),
- you can issue a certificate with multiple DNS names (as in the example above),
- you can validate different domains that are part of the same certificate using different Ingress controllers.

Read more in the cert-manager documentation.

Issuing a DNS wildcard certificate using Cloudflare

1. Get the `Global API Key` and `Email Address`:

- Go to https://dash.cloudflare.com/profile.
- You can find an active `Email Address` at the very top of the page.
- Click the `View` button at the bottom of the page next to the `Global API Key`.

You will see the key for interacting with the Cloudflare API (as well as the account email).

2. Edit the cert-manager module configuration and add the following parameters:

```
settings:
  cloudflareGlobalAPIKey: APIkey
  cloudflareEmail: some@mail.somedomain
```

or

```
settings:
  cloudflareAPIToken: some-token
  cloudflareEmail: some@mail.somedomain
```

After that, Deckhouse will automatically create ClusterIssuer and Secret for Cloudflare in the `d8-cert-manager` namespace.

- Configuration with APIToken is more secure and recommended for use.

3. Create a Certificate with validation via Cloudflare. Note that you must specify `cloudflareGlobalAPIKey` and `cloudflareEmail` in Deckhouse beforehand:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: cloudflare
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
  - "*.domain.com"
```

4. Create an Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  ingressClassName: nginx
  rules:
  - host: "*.domain.com"
    http:
      paths:
      - backend:
          service:
            name: svc-web
            port:
              number: 80
        path: /
  tls:
  - hosts:
    - "*.domain.com"
    secretName: tls-wildcard
```

Issuing a DNS wildcard certificate using Route53

1. Create a user with the appropriate permissions.

- For this, go to the policy management page and create a policy as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "route53:GetChange",
            "Resource": "arn:aws:route53:::change/*"
        },
        {
            "Effect": "Allow",
            "Action": "route53:ChangeResourceRecordSets",
            "Resource": "arn:aws:route53:::hostedzone/*"
        },
        {
            "Effect": "Allow",
            "Action": "route53:ListHostedZonesByName",
            "Resource": "*"
        }
    ]
}
```

- Go to the user management page and create a user with the above policy.

2. Edit the cert-manager module configuration and add the following parameters:

```
settings:
  route53AccessKeyID: <ACCESS_KEY_ID>
  route53SecretAccessKey: <SECRET_ACCESS_KEY>
```

After that, Deckhouse will automatically create ClusterIssuer and Secret for route53 in the `d8-cert-manager` namespace.

3. Create a Certificate with validation via route53. Note that you must specify `route53AccessKeyID` and `route53SecretAccessKey` in Deckhouse beforehand:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: route53
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
  - "*.domain.com"
```

Issuing a DNS wildcard certificate using Google

1. Create a service account with the appropriate role:
   - Go to the policy management page.
   - Select your project.
   - Create a service account with the desired name (e.g., `dns01-solver`).
   - Switch to the service account created.
   - Add a key by clicking the “Add key” button.
   - The `.json` file with the key data will be saved to your computer.
   - Encode the resulting file using the **base64** algorithm:

     ```
     base64 project-209317-556c656b81c4.json
     ```

2. Use the resulting **base-64** string for setting the `cloudDNSServiceAccount` module parameter.

   After that, Deckhouse will automatically create ClusterIssuer and Secret for cloudDNS in the `d8-cert-manager` namespace.

3. Create a Certificate with validation via cloudDNS:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: clouddns
    kind: ClusterIssuer
  dnsNames:
  - "*.domain.com"
```

Issuing a DNS wildcard certificate using Yandex Cloud DNS

To issue a wildcard certificate with DNS-01 validation in Yandex Cloud DNS, configure the module parameters and create a Certificate that references the `yandex` ClusterIssuer.

⚠ If you previously installed the upstream `cert-manager-webhook-yandex` Helm chart manually, uninstall that chart and delete the cluster-scoped APIService `v1alpha1.acme.cloud.yandex.com` before enabling the module settings below. The built-in webhook registers the same APIService name; leaving the manual install in place can break the whole `cert-manager` release or silently break renewals for an existing custom ClusterIssuer.

i The built-in `yandex` ClusterIssuer uses the production Let's Encrypt directory (same as Cloudflare, Route53, and CloudDNS issuers in this module). Confirm that the folder ID, service account role (`dns.editor`), and public DNS zone are correct before creating Certificates, so failed retries do not consume production rate limits. To validate the setup against staging first, create a temporary ClusterIssuer that points at `https://acme-staging-v02.api.letsencrypt.org/directory` and reuses the same webhook solver config.

1. Create a service account in Yandex Cloud and assign the `dns.editor` role (or equivalent) on the folder that contains the public DNS zone.
2. Create an authorized key for the service account and encode the resulting JSON file with **Base64**:

   ```
   base64 authorized_key.json
   ```

3. Set the module parameters:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cert-manager
   spec:
     version: 1
     enabled: true
     settings:
       yandexFolderID: <FOLDER_ID>
       yandexServiceAccountJSON: <AUTHORIZED_KEY_JSON_BASE64>
   ```

   After that, Deckhouse Kubernetes Platform automatically deploys the Yandex Cloud DNS ACME webhook and creates a ClusterIssuer and Secret for Yandex in the `d8-cert-manager` namespace.

   Where:

   - `<FOLDER_ID>`: ID of the Yandex Cloud folder that contains the public DNS zone
   - `<AUTHORIZED_KEY_JSON_BASE64>`: Contents of `authorized_key.json` encoded in Base64

4. Create a Certificate with validation via Yandex Cloud DNS:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: yandex
    kind: ClusterIssuer
  dnsNames:
  - "*.domain.com"
```

Issuing a self-signed certificate

In this case, the entire process is even more straightforward than that of LetsEncrypt. Simply replace the issuer name (`letsencrypt`) with `selfsigned`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com                     # Name of the certificate; you can use it to view the
cert's status.
  namespace: default
spec:
  secretName: example-com-tls             # Name of the secret to store a private key and a
certificate.
  issuerRef:
    kind: ClusterIssuer                  # Link to the ClusterIssuer.
    name: selfsigned
  commonName: example.com                 # Main certificate domain.
  dnsNames:                             # Additional domain certificates. Requires at least a
duplicate commonName record.
  - www.example.com
  - admin.example.com
```

> An example of creating a self-signed certificate manually, without using the `cert-manager`, is available in FAQ.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

What types of certificates are supported?

The module installs the following ClusterIssuers:

- `letsencrypt`
- `letsencrypt-staging`
- `selfsigned`
- `selfsigned-no-trust`

If you need support for other types of certificates, you can add them yourself.

How to add an additional `ClusterIssuer`?

When is an additional `ClusterIssuer` required?

The standard delivery set includes `ClusterIssuer` that issues certificates from the trusted public certificate authority Let's Encrypt or issues self-signed certificates.

To issue certificates for a domain name via Let's Encrypt, the service requires that you verify domain ownership. The `cert-manager` supports several methods for verifying domain ownership when using `ACME` (Automated Certificate Management Environment):

- `HTTP-01` - when using this method `cert-manager` will create a temporary Pod in the cluster that will listen on a specific URL to verify domain ownership. For it to work, you must be able to direct external traffic to this Pod, usually via `Ingress`.
- `DNS-01` - when using this method `cert-manager` makes a TXT record in DNS to verify domain ownership. The `cert-manager` has built-in support for popular DNS providers: AWS Route53, Google Cloud DNS, Cloudflare, and others.

✖ The `HTTP-01` method does not support issuing wildcard certificates.

The `ClusterIssuers` in standard delivery set that issue certificates via Let's Encrypt are divided into two types:

1. `ClusterIssuer` specific to the cloud provider being used.

   Added automatically when filling in the module settings associated with the cloud provider.
   These `ClusterIssuers` support the `DNS-01` method.

   - `clouddns`
   - `cloudflare`
   - `digitalocean`
   - `route53`

2. `ClusterIssuers` using the `HTTP-01` method.

   Added automatically unless their creation is disabled in the module settings.

   - `letsencrypt`
   - `letsencrypt-staging`

In this way, an additional `ClusterIssuer` may be required in the following cases:

1. Certificates need to be issued in a CA other than Let's Encrypt (including a private one). Supported CAs are available in the `cert-manager` documentation
2. Certificates need to be issued via Let's Encrypt using the `DNS-01` method via a third-party provider.

How to add an additional `ClusterIssuer` using Let's Encrypt and `DNS-01` verification method?

To verify domain ownership via Let's Encrypt using the `DNS-01` method, you need to configure the ability to create TXT records in a public DNS.

`cert-manager` has support for mechanisms for creating TXT records in some popular DNS: `AzureDNS`, `Cloudflare`, `Google Cloud DNS`, etc.
The full list is available in the `cert-manager` documentation.

The module automatically creates `ClusterIssuer` of supported cloud providers when filling in the module settings related to the cloud used.
If necessary, you can create such `ClusterIssuer` yourself.

An example of using AWS Route53 is available in the section How to protect `cert-manager` credentials.
The list of all possible `ClusterIssuer`s that can be created is available in the module templates.

Built-in providers configured via module settings: Cloudflare, Amazon Route53, DigitalOcean, Google Cloud DNS, and Yandex Cloud DNS. For Yandex Cloud DNS, follow the steps in Issuing a DNS wildcard certificate using Yandex Cloud DNS.

Using other third-party DNS providers is implemented via the `webhook` method.

When cert-manager makes an `ACME` `DNS-01` call, it sends a request to the webhook server, which then performs the necessary operations to update the DNS record.
When using this method, you need to place a service that will process the hook and create a TXT record in the DNS provider, then create a `ClusterIssuer` with `dns01.webhook` solver settings according to the webhook documentation.

How to add an additional `Issuer` and `ClusterIssuer` using HashiCorp Vault to issue certificates?

You can use this manual for configuring certificate issuance using Vault.

After configuring PKI and enabling Kubernetes authorization, follow these steps:

- Create a `ServiceAccount`:

  ```
  d8 k create serviceaccount issuer
  ```

- Create a `Secret` with a token for this service account. Starting with Kubernetes 1.24, the secret for a `ServiceAccount` is not created automatically:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: issuer
  annotations:
    kubernetes.io/service-account.name: issuer
type: kubernetes.io/service-account-token
EOF
```

- Create an Issuer:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    # HashiCorp instruction has mistype here
    server: http://vault.default.svc.cluster.local:8200
    path: pki/sign/example-dot-com # configure in pki setup step
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: issuer
          key: token
EOF
```

- Create a Certificate resource, to get a TLS certificate, which is issued by Vault CA:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # domains are set on PKI setup
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

How to add `ClusterIssuer` using its own or intermediate CA to to issue certificates?

Follow the steps below to use a custom or interim CA:

- Generate a certificate (if necessary):

```
openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
```

- In the `d8-cert-manager` namespace, create a secret containing certificate file data.

  An example of creating a secret with d8 k:

```
d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --
cert="rootCACert.pem"
```

An example of creating a secret from a YAML file (the contents of the certificate files must be Base64-encoded):

```
apiVersion: v1
data:
  tls.crt: <OUTPUT OF `cat rootCACert.pem | base64 -w0`>
  tls.key: <OUTPUT OF `cat rootCAKey.pem | base64 -w0`>
kind: Secret
metadata:
  name: internal-ca-key-pair
  namespace: d8-cert-manager
type: Opaque
```

You can use any name you like for the secret.

- Create a `ClusterIssuer` using the secret you created earlier:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: inter-ca
spec:
  ca:
    secretName: internal-ca-key-pair   # Name of the secret you created earlier.
```

You can use any name as your `ClusterIssuer` name.

You can now use the created `ClusterIssuer` to issue certificates for all Deckhouse components or a particular component.

For example, to issue certificates for all Deckhouse components, specify the `ClusterIssuer` name in the ClusterIssuerName global parameter (`d8 k edit mc global`):

```
spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
        mode: CertManager
      publicDomainTemplate: '%s.<public_domain_template>'
  version: 1
```

How does cert-manager work behind a proxy server?

`cert-manager` uses the cluster-wide proxy server set in the proxy parameter of the ClusterConfiguration resource. The proxy settings are passed to the `cert-manager` controller and to the Yandex Cloud DNS webhook. Requests to the ACME server, to HashiCorp Vault and to the HTTP APIs of DNS providers go through the proxy server.

The following requests go directly:

- requests to the cluster domain, to the pod and service subnets;
- requests of the controller to service names in the `<SERVICE>.<NAMESPACE>.svc` form;
- requests to the addresses listed in `noProxy`.

Node addresses and short service names, such as `<SERVICE>.<NAMESPACE>`, are not on this list. To reach an issuer inside the cluster directly, address it by the full `<SERVICE>.<NAMESPACE>.svc` name. To reach an issuer in the internal network directly, add its address to `noProxy`. Keep in mind that `noProxy` also applies to the other cluster components and to the nodes.

Before the `DNS-01` verification, the controller checks the TXT record over the DNS protocol, which a proxy server does not carry. If the cluster has no direct access to external DNS servers, set DNS over HTTPS in the recursiveSettings parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  version: 1
  settings:
    recursiveSettings:
      nameservers: ["https://1.1.1.1/dns-query"]
      useOnlyRecursive: true
```

With `useOnlyRecursive: true`, the controller sends the checks only to the listed servers, and DNS over HTTPS requests go through the proxy server. A DNS server of the internal network that resolves external names works as well: specify it in the `IP:port` form.

Before the `HTTP-01` verification, the controller also requests the verification URL of the domain through the proxy server. If the proxy server cannot reach the public address of the cluster, add the domain to `noProxy`.

How to secure `cert-manager` credentials?

If you don’t want to store credentials in the Deckhouse configuration (security reasons, for example), feel free to create your own ClusterIssuer / Issuer. For example, you can create your own `ClusterIssuer` for a route53 service in this way:

- Create a Secret with credentials:

  ```
  d8 k apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  type: Opaque
  metadata:
    name: route53
    namespace: default
  data:
    secret-access-key: MY-AWS-ACCESS-KEY-TOKEN
  EOF
  ```

- Create a simple `ClusterIssuer` with reference to that secret:

  ```
  d8 k apply -f - <<EOF
  apiVersion: cert-manager.io/v1
  kind: ClusterIssuer
  metadata:
    name: route53
    namespace: default
  spec:
    acme:
      server: https://acme-v02.api.letsencrypt.org/directory
      privateKeySecretRef:
        name: route53-tls-key
      solvers:
      - dns01:
          route53:
            region: us-east-1
            accessKeyID: MY-AWS-ACCESS-KEY-ID
            secretAccessKeySecretRef:
              name: route53
              key: secret-access-key
  EOF
  ```

- Order certificates as usual, using created `ClusterIssuer`:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: route53
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

Does the tls-acme annotation work?

Yes, it works! The dedicated component (`cert-manager-ingress-shim`) automatically creates `Certificate` resources based on these annotations (in the same namespaces as those of Ingress resources with annotations).

i The Certificate for a particular annotation is linked to the existing Ingress resource. The additional records are put into the existing Ingress resource instead of creating a separate one. Thus, the process will fail if authentication or whitelist is set for the primary Ingress. In this case, you shouldn’t use the annotation; use the Certificate instead.

⚠ If you switched to the Certificate instead of annotation, delete the annotation-based Certificate. Otherwise, the same Secret will be updated for both Certificates, and this may lead to exceeding the Let’s Encrypt limits.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/tls-acme: "true"         # The annotation.
  name: example-com
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  - host: www.example.com                # The additional domain
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  - host: admin.example.com               # Another additional domain
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - example.com
    - www.example.com                   # The additional domain
    - admin.example.com                  # Another additional domain
    secretName: example-com-tls           # The name of the certificate and secret
```

How do I check the certificate status?

```
d8 k -n default describe certificate example-com
...
Status:
  Acme:
    Authorizations:
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   example.com
      Uri:      https://acme-
v01.api.letsencrypt.org/acme/challenge/qJA9MGCZnUnVjAgxhoxONvDnKAsPatRILJ4n0lJ7MMY/4062050823
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   admin.example.com
      Uri:      https://acme-v01.api.letsencrypt.org/acme/challenge/pW2tFKLBDTll2Gx8UBqmEl846x5W-
YpBs8a4HqstJK8/4062050808
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   www.example.com
      Uri:      https://acme-v01.api.letsencrypt.org/acme/challenge/LaZJMM9_OKcTYbEThjT3oLtwgpkNfbHVdl8Dz-
yypx8/4062050792
  Conditions:
    Last Transition Time:  2018-04-02T18:01:04Z
    Message:               Certificate issued successfully
    Reason:                CertIssueSuccess
    Status:                True
    Type:                  Ready
Events:
  Type     Reason                 Age                From                     Message
  ----     ------                 ----               ----                     -------
  Normal   PrepareCertificate     1m                 cert-manager-controller  Preparing certificate with issuer
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain example.com
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain www.example.com
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain admin.example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
admin.example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
www.example.com
  Normal   ObtainAuthorization    55s                cert-manager-controller  Obtained authorization for domain
example.com
  Normal   ObtainAuthorization    54s                cert-manager-controller  Obtained authorization for domain
admin.example.com
  Normal   ObtainAuthorization    53s                cert-manager-controller  Obtained authorization for domain
www.example.com
```

How do I get a list of certificates?

```
d8 k get certificate --all-namespaces
NAMESPACE          NAME                         AGE
default            example-com                  13m
```

The “CAA record does not match issuer” error

Suppose `cert-manager` gets the following error when trying to provide a certificate:

```
CAA record does not match issuer
```

In this case, you have to check the `CAA (Certificate Authority Authorization)` DNS record of the domain for which the certificate is intended. For Let’s Encrypt certificates, the domain must have the `issue "letsencrypt.org"` CAA record. You can read more about CAA in the Let’s Encrypt documentation.

What to do if the HTTP-01 self-check never completes?

Before cert-manager asks Let’s Encrypt to validate an `HTTP-01` Challenge, it requests the challenge URL of the domain from inside the cluster. If the cluster cannot reach its own public address, this self-check never succeeds, and the Challenge stays pending, although Let’s Encrypt could validate it. It happens, for example, behind NAT without hairpinning (when you cannot access your external IP address from inside the cluster) or with split-horizon DNS (when the same domain resolves to different IP addresses depending on where the DNS request originated: from inside the cluster or from outside).

The Challenge then reports the self-check error. To see it, run the following command:

```
d8 k -n <NAMESPACE> describe challenges.acme.cert-manager.io
```

To skip the self-check for the `letsencrypt` and `letsencrypt-staging` ClusterIssuers, set the

`http01WaitInsteadOfSelfCheck` parameter. The following ModuleConfig makes cert-manager wait 30 seconds instead of the self-check:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  version: 1
  settings:
    http01WaitInsteadOfSelfCheck: 30s
```

After the ClusterIssuers are updated, a new Challenge no longer waits for the successful self-check. cert-manager asks Let's Encrypt to validate it 30 seconds after the challenge is presented.

Use the parameter only when the domain is reachable from the internet. Otherwise every validation attempt fails on the side of Let's Encrypt and counts against its rate limits.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The chrony module

> **i** The module lifecycle stage: General Availability

Provides time synchronization on all cluster nodes using the chrony utility.

How it works

The module starts `chrony` agents on all cluster nodes.

By default, the NTP server `pool.ntp.org` is used. The NTP server can be changed via the settings module.

To view the NTP servers used, you can use the command:

```
d8 k exec -it -n d8-chrony ds/chrony-master -- chronyc -N sources
```

Ouput example:

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* pool.ntp.org.                 2  10   377   171   -502us[ -909us] +/- 5388us
^- pool.ntp.org.                 2  10   377   666  -5317us[-5698us] +/-  103ms
^+ pool.ntp.org.                 2  10   377   938   -201us[ -567us] +/- 5346us
^+ pool.ntp.org.                 2  10   377   843   -159us[ -530us] +/-   12ms
```

> **i** `^+` - combined NTP server (`chrony` combines information from `combined` servers to reduce inaccuracies);
> `^*` - current NTP server;
> `^-` - non-combinable NTP server.
> Extra flag `-v` adds information about meanings of the columns: `chronyc sources -v`.

`chrony` agents on master nodes and on other nodes have one main difference - on all nodes that are not masters, the list of NTP servers contains not only NTP servers from `module config`, but also the addresses of all master nodes of the cluster.

Thus, agents on master nodes synchronize time only from the list of hosts specified in `module config` (by default from `pool.ntp.org`). And agents on other nodes synchronize time with the list of NTP servers from `module config` plus with `chrony` agents on master nodes.

This is done so that in case of unavailability of NTP servers specified in `module config`, the time is synchronized with the master nodes.

Boot-time synchronization on master nodes

On master nodes, the module also sets the clock once during boot, before `kubelet` starts. This keeps control plane static Pods from starting with a wrong time taken from the real-time clock (RTC) or the hypervisor.

If most of the NTP servers that respond agree on the time within 1 second, `kubelet` starts right after the clock is set. If no NTP server responds, or the servers that respond disagree, `kubelet` starts with an unverified clock after up to 30 seconds. On nodes managed by bashible, the limit can be raised to 60 seconds using `earlyTimeSync.timeoutSeconds`. On nodes managed by bashible, the time spent waiting for the node network is not included. On these nodes, systemd waits for `nodelet.service` to report readiness after the check. The check runs in the main nodelet process. On Deckhouse Engine nodes, the wait starts when nodelet starts and can include the wait for the node network. Boot-time synchronization can be turned off with the `earlyTimeSync.enabled` parameter.

Boot-time synchronization uses the node network and the node DNS settings. Kubernetes DNS and NTP servers running inside the cluster are not available at this point.

If a master node with boot-time synchronization configured cannot reach the NTP servers, the check delays `kubelet` up to its limit. The alert fires once monitoring collects the failed check. This applies to an air-gapped cluster that keeps the default `pool.ntp.org` server. In that case, set `ntpServers` to NTP servers that the master nodes can reach, or disable boot-time synchronization.

Enabling boot-time synchronization or changing its parameters takes effect on the next boot. The clock on a running node is not changed immediately, because stepping it back on a live control plane can disrupt authentication and leader election. Disabling boot-time synchronization with `earlyTimeSync.enabled: false` takes effect as soon as the configuration is applied on the master nodes.

If the clock was not synchronized during boot, the `NodeTimeNotSynchronizedAtBoot` alert fires for the master node. The alert stays active until one of the following happens:

- The node boots again and the synchronization succeeds.
- Boot-time synchronization is disabled. On Deckhouse Engine nodes, the alert then clears on the next boot.
- You clear the alert as described in the alert description.

The alert works only when the `monitoring-kubernetes` module is enabled, because that module collects the result from the node.

Limitations

Boot-time synchronization has the following limitations:

- On Deckhouse Engine nodes, `earlyTimeSync.timeoutSeconds` does not apply: `kubelet` waits for 30 seconds at most. Only the first 16 host names and IPv4 addresses of `ntpServers` are used there. When a master node boots into a new OS image, this wait counts toward the 15 minutes the node has to confirm the image. If the NTP servers are unreachable, set reachable servers in `ntpServers` or disable boot-time synchronization.
- Values of `ntpServers` other than a host name or an IPv4 address, such as an IPv6 address or a value with a port, are skipped.
- It does not cover the first boot of a new master node, including the cluster bootstrap. The configuration reaches the node during that boot, after `kubelet` has already started.
- On nodes managed by bashible, disabling the module leaves the boot-time synchronization files on the nodes. Deckhouse Engine nodes stop the synchronization on their own from the next boot. Before disabling the module, set `earlyTimeSync.enabled` to `false` and wait until the configuration is applied on the master nodes. If the module is already disabled, remove the files with a NodeGroupConfiguration as shown in the FAQ.

Third-party components

List of third-party software used in the `chrony` module:

- Chrony

  License: GNU General Public License version 2

  chrony is a versatile implementation of the Network Time Protocol (NTP)

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** The module lifecycle stage: General Availability

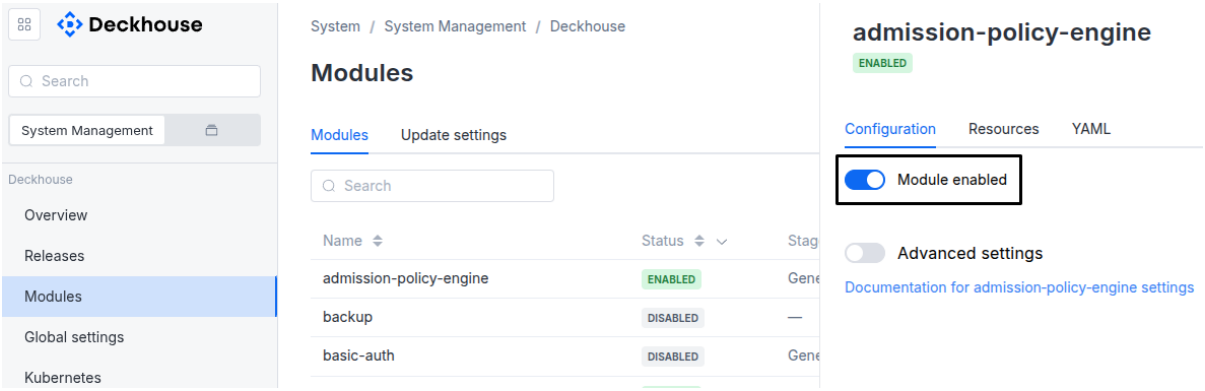
The module has 3 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `chrony` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable chrony
```

- **Using ModuleConfig `chrony`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `chrony` (create it if necessary);

  Example of a manifest to enable module `chrony`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
```

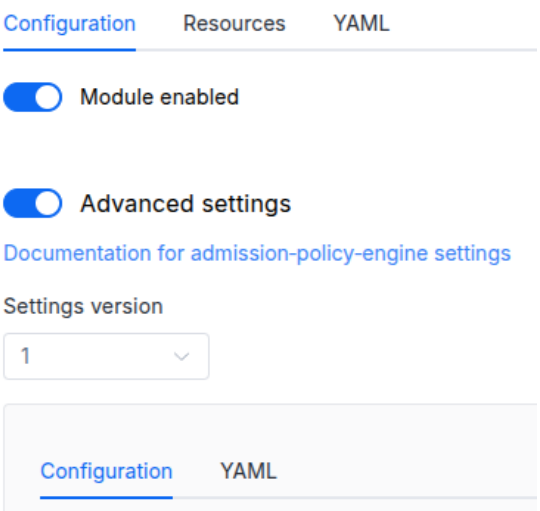
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `chrony` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `chrony` on the “YAML” tab in the module settings window

("System" → "System Management" → "Deckhouse" → "Modules", open the module `chrony`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `chrony` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc chrony
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `chrony` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `chrony`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: chrony
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.earlyTimeSync**
    object

    Time synchronization on master nodes during boot, before kubelet starts.

    It prevents control plane static Pods from starting with an incorrect time received from the real-time clock (RTC) or the hypervisor. Enabling and changing the settings take effect on the next boot of the node. Disabling takes effect as soon as the configuration is applied.

    - **settings.earlyTimeSync.enabled**
      boolean

      Enables boot-time synchronization, before kubelet starts.

      Default: `true`

    - **settings.earlyTimeSync.timeoutSeconds**
      integer

      Maximum time to wait for synchronization before kubelet starts, in seconds.

      On nodes managed by bashible, the wait starts once the node network is available. If within this time no NTP server responds, or the servers that respond disagree, kubelet starts with an unverified clock. On Deckhouse Engine nodes, the parameter does not apply: the wait is at most 30 seconds.

      If a master node already configured for boot-time synchronization cannot reach the NTP servers, for example in an air-gapped cluster with the default `pool.ntp.org`, the check waits for the full time. Set reachable servers in ntpServers or disable boot-time synchronization.

      Default: `30`

      Allowed values: `30 <= X <= 60`

  - **settings.ntpServers**
    array of strings

    NTP servers used for time synchronization.

Each server is specified as a host name or an IPv4 address, without a port.

On master nodes, these servers are also used for boot-time synchronization, before kubelet starts. On Deckhouse Engine nodes, boot-time synchronization uses the first 16 of them. Boot-time synchronization is configured by earlyTimeSync. An empty list disables boot-time synchronization.

Default: `["pool.ntp.org"]`

Examples:

```
ntpServers:
- pool.ntp.org
```

```
ntpServers:
- pool.ntp.org
- ntp.ubuntu.com
- time.google.com
```

An example of the configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - pool.ntp.org
      - ntp.ubuntu.com
      - time.google.com
  version: 1
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How do I disable chrony and use NTP daemon on nodes?

1. Disable usage of chrony module.
2. Create `NodeGroupConfiguration` custom step to enable use NTP daemon on nodes (example for `systemd-timesyncd`):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: enable-ntp-on-node.sh
spec:
  weight: 100
  nodeGroups: ["*"]
  bundles: ["*"]
  content: |
    if [ -f /etc/systemd/system/kubelet.service.d/20-d8-early-time-sync.conf ] || [ -f
/etc/systemd/system/nodelet.service.d/20-d8-early-time-sync.conf ]; then
      rm -f /etc/systemd/system/kubelet.service.d/20-d8-early-time-sync.conf
      rm -f /etc/systemd/system/nodelet.service.d/20-d8-early-time-sync.conf
      systemctl daemon-reload
    fi
    rm -f /var/run/node-exporter-textfile/d8_early_time_sync.prom
    systemctl enable systemd-timesyncd
    systemctl start systemd-timesyncd
```

The script removes the boot-time synchronization drop-ins and metric file that remain after disabling the module.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The cilium-hubble module

The module lifecycle stage: General Availability

The module has requirements for installation

Web interfaces associated with the module: hubble

The `cilium-hubble` module provides visualization of the cluster network stack if the cilium CNI is enabled.

Requirements

The following is required for the `cilium-hubble` module:

- Linux kernel version >= 5.8 with eBPF support.
- BPF Type Format (BTF) support enabled. You can verify it as follows:
  - Run `ls -lah /sys/kernel/btf/vmlinux` — if the file exists, BTF is supported.
  - Run `grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*` — if the parameter is enabled, BTF is supported.

Third-party components

List of third-party software used in the `cilium-hubble` module:

- Hubble UI 0.13.5

  License: Apache License 2.0

  Hubble UI is an open-source user interface for Cilium Hubble.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** The module lifecycle stage: General Availability
>
> The module has requirements for installation

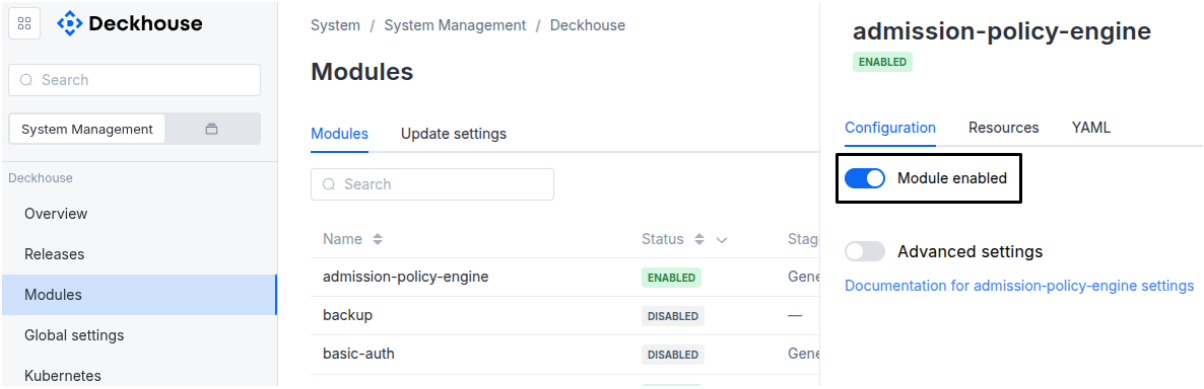
The module is not enabled by default in any bundles.

If the `cni-cilium` module is disabled, the `ciliumHubbleEnabled:` parameter will not affect the enabling of the `cilium-hubble` module.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cilium-hubble` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cilium-hubble
```

- **Using ModuleConfig `cilium-hubble`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cilium-hubble` (create it if necessary);

  Example of a manifest to enable module `cilium-hubble`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cilium-hubble
spec:
  enabled: true
```

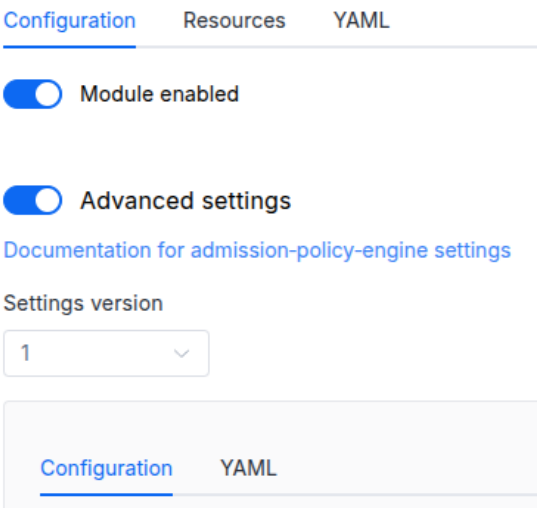
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cilium-hubble` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cilium-hubble` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cilium-hubble`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cilium-hubble` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cilium-hubble
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cilium-hubble` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cilium-hubble`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cilium-hubble
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `cni-cilium`: any version.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

  - `del(.auth.password) | if .auth == {} then del(.auth) end`

Parameters

Schema version: 2

- **settings**
  object
  - **settings.auth**
    object

    Options related to authentication or authorization in the Hubble web UI.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module's public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access Hubble web UI.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

      Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

      > **i** External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

        Example:

        ```
        authSignInURL: https://example.com/dex/sign_in
        ```

      - **settings.auth.externalAuthentication.authURL**
        string

        The URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

        Example:

        ```
        authURL: https://example.com/dex/auth
        ```

    - **settings.auth.whitelistSourceRanges**
      array of strings

      An array if CIDRs that are allowed to authenticate in Hubble web UI.

      Example:

      ```
      whitelistSourceRanges:
      - 1.1.1.1/32
      ```

  - **settings.debugLogging**
    boolean

    Enabled debug logging for Cilium Hubble component.

    Default: `false`

  - **settings.https**
    object

    What certificate type to use.

This parameter completely overrides the `global.modules.https` settings.

Examples:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**
  object

  Parameters for certmanager.

  - **settings.https.certManager.clusterIssuerName**
    string

    What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

    Default: `letsencrypt`

    Examples:

    ```
    clusterIssuerName: letsencrypt
    ```

    ```
    clusterIssuerName: letsencrypt-staging
    ```

    ```
    clusterIssuerName: selfsigned
    ```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the Hubble web UI.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter;
  - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace;
  - `Disabled` — in this mode, the documentation web UI can only be accessed over HTTP;
  - `OnlyInURI` — the documentation web UI will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for Hubble.

Optional. By default, the `modules.ingressClass` global value is used.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.tolerations**
  array of objects

  The same as `spec.tolerations` for the Kubernetes Pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured. If these options are disabled, the module will use basic auth with the auto-generated password.

To view the generated password, run the command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values cilium-hubble -o json | jq '.ciliumHubble.internal.auth.password'
```

To generate a new password, delete the Secret:

```
d8 k -n d8-cni-cilium delete secret/hubble-basic-auth
```

> **i** The `auth.password` parameter is deprecated.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — AWS

> **i** The module lifecycle stage: General Availability

The `cloud-provider-aws` module integrates Deckhouse Platform with Amazon AWS. It allows the `node-manager` module to use AWS resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-aws` module:

- Managing AWS resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the AWS side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in AWS.
- Provisioning disks via the EBS CSI driver (`ebs.csi.aws.com`) and creating StorageClasses for AWS volume types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-aws`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the AWSInstanceClass resource.
- Registering with `node-manager` so that AWSInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Third-party components

List of third-party software used in the `cloud-provider-aws` module:

- Amazon Web Services Cloud provider v1.33.4, v1.34.3, v1.35.2, v1.36.0, v1.37.0

  License: Apache License 2.0

  Cloud Controller Manager for Amazon Web Services.

- Amazon Web Services CSI driver v1.54.0, v1.54.0, v1.54.0, v1.54.0, v1.66.0

  License: Apache License 2.0

  CSI driver for Amazon EBS.

- Amazon Web Services Terraform provider 6.69.1

  License: Mozilla Public License 2.0

  Terraform provider for Amazon Web Services.

- Amazon Web Services Node termination handler v1.25.6

  License: Apache License 2.0

  Gracefully handle instance shutdown within Kubernetes for Amazon EC2.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy (the AWSClusterConfiguration custom resource). In most cases, you do not have to configure the module manually.

You can configure the number and parameters of provisioning machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the `cloudInstances.ClassReference` NodeGroup parameter). In the case of the AWS cloud provider, the instance class is the `AWSInstanceClass` custom resource that stores specific parameters of the machines.

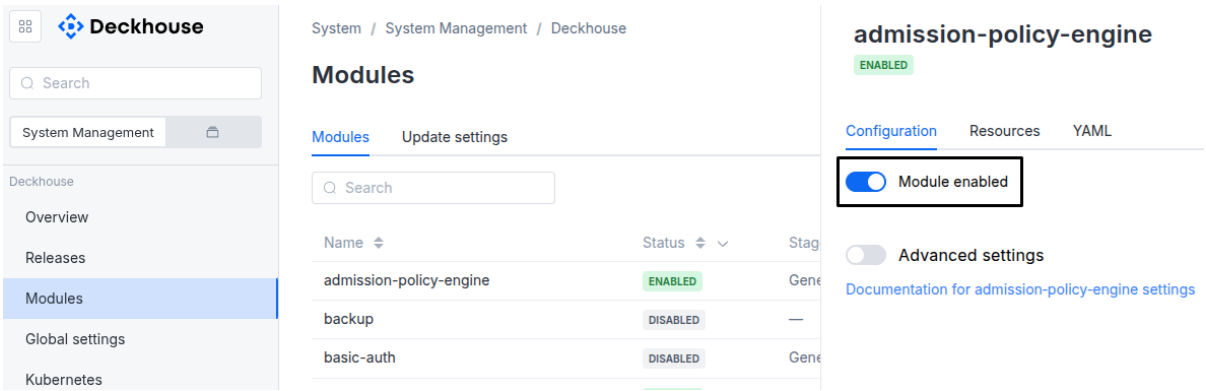
The module automatically creates StorageClasses that are available in AWS: `gp3`, `gp2`, `sc1`, and `st1`. It lets you configure disks with the required IOPS. Also, it can filter out the unnecessary StorageClasses (you can do this via the `exclude` parameter).

The module has 1 alert.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `cloud-provider-aws` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable cloud-provider-aws
```

- **Using ModuleConfig `cloud-provider-aws`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-aws` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-aws`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-aws
spec:
  enabled: true
```

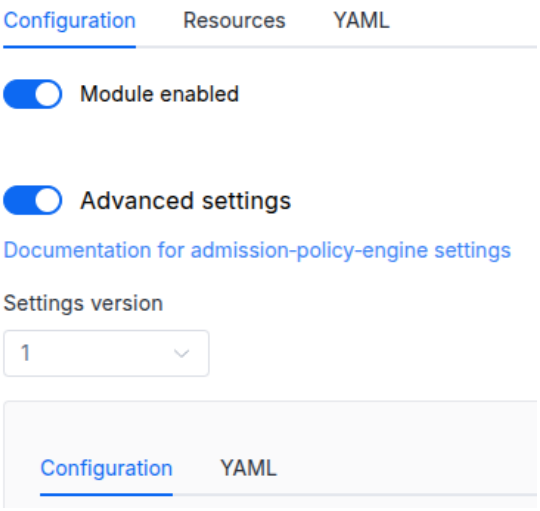
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-aws` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cloud-provider-aws` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-aws`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-aws` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-aws
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-aws` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-aws`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-aws
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - an arbitrary StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: gp3
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

      Example:

      ```
      exclude:
      - sc.*
      - st1
      ```

    - **settings.storageClass.provision**
      array of objects

      Defines additional StorageClasses or override the parameters of created by default.

      - **settings.storageClass.provision.iops**
        string

        I/O operations per second for `gp3` volume type:

        - Allowed values: from `3000` to `16000`.
        - If empty, it will set to 3000 as documented here.
        - Changing doesn't apply to existing volumes. You can modify them manually through AWS Console.
      - **settings.storageClass.provision.iopsPerGB**
        string

        The number of I/O operations per second per GB for `io1` and `io2` volume types (this parameter is `3` for `gp2` volumes):

        - **Caution!** If the iopsPerGB value multiplied by the target volume's size is less than 100 or more than 64000, the creation of such a volume will fail.
        - You can find a detailed description of the volume types and their IOPS in the official documentation.
      - **settings.storageClass.provision.name**
        string

Required value

The name of the class to create.

- **settings.storageClass.provision.throughput**
  string

  Throughput in MiB/s for `gp3` volume type:

  - Allowed values: from `125` to `1000`.
  - If empty, it will set to 125 as documented here.
  - Changing doesn't apply to existing volumes. You can modify them manually through AWS Console.
- **settings.storageClass.provision.type**
  string

  Required value

  The volume type.

  Allowed values: `gp3`, `gp2`, `sc1`, `st1`, `io1`, `io2`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

AWSClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in AWS.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 172.16.0.0/22
vpcNetworkCIDR: 172.16.0.0/16
masterNodeGroup:
  replicas: 1
  instanceClass:
    instanceType: m5.xlarge
    ami: ami-08b6d44b4f6f7b279
    diskType: gp3
nodeGroups:
- name: worker
  nodeTemplate:
    labels:
      node-role.kubernetes.io/worker: ''
  replicas: 2
  instanceClass:
    instanceType: t2.medium
    ami: ami-0aad10862ade98f27
  additionalTags:
    backup: srv1
provider:
  providerAccessKeyId: "<AWS_ACCESS_KEY>"
  providerSecretAccessKey: "<AWS_SECRET_ACCESS_KEY>"
  region: eu-central-1
tags:
  team: rangers
```

- **additionalRolePolicies**
  array of strings

  A list containing additional policy actions for IAM roles.

  - Additional policy actions would be attached to the default IAM role policy actions.
  - Parameter is optional. If omitted, only default IAM role policy actions are used.
  - Example of policy actions: `ecr:ListImages`, `s3:GetObject`, etc.

  Default IAM role policies actions contain the following roles:

  - `ec2:DescribeTags`
  - `ec2:DescribeInstances`
  - **Element of the array**
    string

    Pattern: `^([a-zA-Z0-9_\-]+):([a-zA-Z0-9_\-\*\?]+)$`

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **disableDefaultSecurityGroup**
  boolean

  If set to `true`, the default security group will not be created.

  Warning. When using `disableDefaultSecurityGroup: true`, you must manually create all required security groups to allow access to cluster nodes. Additionally, you must explicitly specify them in the following parameters:

  - `additionalSecurityGroups` in the `masterNodeGroup` section of the `AWSClusterConfiguration` resource
  - `additionalSecurityGroups` in the `AWSInstanceClass` resource
  - `additionalSecurityGroups` in the `nodeGroups.instanceClass` section

  To configure the security groups used by load balancers, specify them using the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

  For details, see AWS: Layouts.

- **existingVPCID**
  string

ID of the existing VPC to use for deploying.

  - **A mandatory parameter** if the `vpcNetworkCIDR` is omitted.
  - **Caution!** If there is an Internet Gateway in the target VPC, the deployment of the basic infrastructure will fail with an error. Currently, an Internet Gateway cannot be adopted.
- **iamNodeRole**
string

The name of the IAM role that will be attached to all AWS instances of the cluster nodes.

DP always creates and assigns a special IAM role named `<PREFIX>-node` to each AWS instance of the cluster nodes (`<PREFIX>` is the value of the `prefix` parameter in the ModuleConfig `global`). You can specify your IAM-role with greater permissions in the `iamNodeRole` parameter, but it is important that it includes the IAM policies of the role created by DP (the `<PREFIX>-node` role).

More details about IAM roles for AWS EC2 can be found in the AWS documentation.

Pattern: `^[a-zA-Z0-9+=,.@\-_']{1,64}$`

- **imdsv2**
boolean

Enables mandatory use of IMDSv2 and disables IMDSv1 for EC2 instances in the cluster.

Using IMDSv2 is recommended. IMDSv2 requires a session token to access instance metadata, which reduces the risk of unauthorized access, including through SSRF (Server-Side Request Forgery) attacks.

For backward compatibility, the default value of the parameter is `false`. However, if the parameter is not specified in an existing configuration, DP does not change the current instance metadata service settings. Once the parameter is explicitly specified, the IMDS settings are managed by DP:

  - `false` — allows both IMDSv1 and IMDSv2
  - `true` — allows only IMDSv2 and requires a session token to access instance metadata

> **i** **Warning.** When IMDSv2 is enabled, DP sets the IMDS hop limit to `1`. If the AWS account defaults or the AMI provide a higher value, it is lowered to `1` on the next converge. As a result, the instance metadata service is unavailable from pods, even if the applications running in those pods support IMDSv2. This limitation does not affect DP components because they use the host network. Before enabling IMDSv2, make sure that applications requiring access to instance metadata support IMDSv2 and run using the host network.

To apply the change to master, static, and bastion nodes, run `dhctl converge`. CloudEphemeral nodes are recreated automatically.

Default: `false`

- **kind**
string

Required value

Allowed values: `AWSClusterConfiguration`

- **layout**
string

Required value

The way resources are located in the cloud.

Read more about possible provider layouts.

> **i** **Note,** that the `Standard` layout is deprecated.

Allowed values: `WithoutNAT`, `WithNAT`, `Standard`

- **masterNodeGroup**
object

Required value

Parameters of the master's NodeGroup.

> **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.additionalTags**
  object

The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

Example:

```
additionalTags:
  project: cms-production
  severity: critical
```

- **masterNodeGroup.instanceClass**
  object

  Required value

  Partial contents of the fields of the AWSInstanceClass.

  - **masterNodeGroup.instanceClass.additionalSecurityGroups**
    array of strings

    The additional security groups to add to provisioned instances of the specific InstanceClass.

  - **masterNodeGroup.instanceClass.ami**
    string

    Required value

    The Amazon Machine Image (AMI ID) to use in provisioned instances.

    Here is how you can find the required AMI (each region has its own set of AMIs):

    ```
    aws ec2 --region <REGION> describe-images \
    --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
    jq '.Images[].ImageId'
    ```

    Example:

    ```
    ami: ami-040a1551f9c9d11ad
    ```

  - **masterNodeGroup.instanceClass.diskSizeGb**
    integer

    Instance disk size in gibibytes.

    Example:

    ```
    diskSizeGb: 50
    ```

  - **masterNodeGroup.instanceClass.diskType**
    string

    Instance EBS disk type.

    Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

    Example:

    ```
    diskType: gp2
    ```

  - **masterNodeGroup.instanceClass.etcdDisk**
    object

    Default: `{"sizeGb":20,"type":"gp3"}`

    - **masterNodeGroup.instanceClass.etcdDisk.sizeGb**
      integer

      Etcd disk size in gibibytes.

    - **masterNodeGroup.instanceClass.etcdDisk.type**
      string

      Etcd disk type.

  - **masterNodeGroup.instanceClass.instanceType**
    string

    Required value

Instance type of AWS instance.

Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

Example:

```
instanceType: t3.large
```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  Not required value.

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    project: cms-production
    severity: critical
    ```

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the AWSInstanceClass.

    - **nodeGroups.instanceClass.additionalSecurityGroups**
      array of strings

      The additional security groups to add to provisioned instances of the specific InstanceClass.

    - **nodeGroups.instanceClass.ami**
      string

      Required value

      The Amazon Machine Image (AMI ID) to use in provisioned instances.

      Here is how you can find the required AMI (each region has its own set of AMIs):

      ```
      aws ec2 --region <REGION> describe-images \
      --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-
      server-2020*' | \
      jq '.Images[].ImageId'
      ```

      Example:

      ```
      ami: ami-040a1551f9c9d11ad
      ```

    - **nodeGroups.instanceClass.diskSizeGb**
      integer

      Instance disk size in gibibytes.

Example:

```
diskSizeGb: 50
```

- **nodeGroups.instanceClass.diskType**
  string

  Instance EBS disk type.

  Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

  Example:

  ```
  diskType: gp2
  ```

- **nodeGroups.instanceClass.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

  ```
  instanceType: t3.large
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup. It is used to generate the node name.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

        - **nodeGroups.nodeTemplate.taints.effect**
          string

          Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
        - **nodeGroups.nodeTemplate.taints.key**
          string
        - **nodeGroups.nodeTemplate.taints.value**
          string
    - **nodeGroups.replicas**
      integer

      Required value

      The number of nodes.
    - **nodeGroups.zones**
      array of strings

      Not required value.

      A limited set of zones in which nodes can be created.

      The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeNetworkCIDR**
  string

  A subnet to use for cluster nodes:

  - The IP range must overlap or match the VPC address range.
  - The IP range will be evenly split into subnets, one per Availability Zone in your region.
  - An optional but recommended parameter. By default, it corresponds to the whole range of VPC addresses.

  If a new VPC is created along with a new cluster and no `vpcNetworkCIDR` is provided, then the range from `nodeNetworkCIDR` is used for the VPC. Thus, the entire VPC is allocated for the cluster networks, and you will not be able to add other resources to this VPC.

  The `nodeNetworkCIDR` range is distributed between subnets depending on the number of availability zones in the selected region. For example, if `nodeNetworkCIDR: "10.241.1.0/20"` and there are three availability zones in the region, subnets will be created with the `/22` mask.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **peeredVPCs**
  array of strings

  A list of AWS VPC IDs to peer with the cluster network.

  The service account must have access to all the VPCs listed. You have to configure the peering connection manually if no access is available.

- **provider**
  object

  Required value

  Contains settings to connect to the AWS API.

  - **provider.providerAccessKeyId**
    string

    Required value

    Access key ID.
  - **provider.providerSecretAccessKey**
    string

    Required value

    Access key secret.

    - **provider.region**
      string

      Required value

      The name of the AWS region where instances will be provisioned.

- **publicNetworkAllowList**
  array of strings

  A list of CIDRs allowed to access public endpoints of the cluster.

  Caution. AWS limits the number of inbound and outbound rules per security group to 60 each. If you need to specify more CIDRs, request a quota increase via AWS Support.

  Default: `["0.0.0.0/0"]`

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, `0.0.0.0/0`.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  Deprecated
  object

  Layout is **deprecated**.

- **tags**
  object

  Not required value.

  A dictionary of tags to create on all resources that support this feature.

  You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **vpcNetworkCIDR**
  string

  A subnet to use in the VPC being created.

  A mandatory parameter if the `existingVPCID` parameter is omitted.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withNAT**
  object
    - **withNAT.bastionInstance**
      object
        - **withNAT.bastionInstance.instanceClass**
          object

          Required value

          Partial contents of the fields of the AWSInstanceClass.

            - **withNAT.bastionInstance.instanceClass.additionalSecurityGroups**
              array of strings

              The additional security groups to add to provisioned instance of the specific InstanceClass.

            - **withNAT.bastionInstance.instanceClass.ami**
              string

              Required value

              The Amazon Machine Image (AMI ID) to use in provisioned instance.

              Here is how you can find the required AMI (each region has its own set of AMIs):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
jq '.Images[].ImageId'
```

Example:

```
ami: ami-040a1551f9c9d11ad
```

- **withNAT.bastionInstance.instanceClass.diskSizeGb**
  integer

  Instance disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 20
  ```

- **withNAT.bastionInstance.instanceClass.diskType**
  string

  Instance EBS disk type.

  Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

  Example:

  ```
  diskType: gp2
  ```

- **withNAT.bastionInstance.instanceClass.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in selected zone.

  Example:

  ```
  instanceType: t3.large
  ```

- **withNAT.bastionInstance.zone**
  string

  The zone in which the bastion instance will be created.

  By default, the first available zone in the region or the first from the list of the global parameter `zones` will be used.

- **withoutNAT**
  object
- **zones**
  array of strings

  The globally restricted set of zones that this cloud provider works with.

  Restricts only the **creation** of nodes in the cluster.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> The module lifecycle stage: General Availability

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

In Amazon AWS, Ubuntu distributions with a custom kernel without `erofs` are used. To bring the OS into compliance with the requirements and add the necessary modules, you can use the command `sudo apt install linux-modules-extra-aws`.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

To use the `cloud-provider` and `machine-controller-manager` modules, you must access the AWS API as an IAM user with a sufficient set of privileges.

⚠ Make sure that you have access to the desired regions and that you have the necessary quotas.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

JSON Policy

First, prepare a JSON file with the configuration of the necessary privileges:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:AttachVolume",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateInternetGateway",
                "ec2:CreateKeyPair",
                "ec2:CreateNATGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateVpc",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteKeyPair",
                "ec2:DeleteNATGateway",
                "ec2:DeleteRoute",
```

```
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DeleteVpc",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAddressesAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeInstanceTopology",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRegions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVolumesModifications",
                "ec2:DescribeVolumes",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcClassicLink",
                "ec2:DescribeVpcClassicLinkDnsSupport",
                "ec2:DescribeVpcs",
                "ec2:DetachInternetGateway",
                "ec2:DetachVolume",
                "ec2:DisassociateAddress",
                "ec2:DisassociateRouteTable",
                "ec2:ImportKeyPair",
                "ec2:ModifyInstanceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVolume",
                "ec2:ModifyVpcAttribute",
                "ec2:ReleaseAddress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:CreateVpcPeeringConnection",
                "ec2:DeleteVpcPeeringConnection",
                "ec2:AcceptVpcPeeringConnection",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "elasticloadbalancing:AddTags",
                "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
                "elasticloadbalancing:AttachLoadBalancerToSubnets",
                "elasticloadbalancing:ConfigureHealthCheck",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateLoadBalancerListeners",
                "elasticloadbalancing:CreateLoadBalancerPolicy",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteLoadBalancerListeners",
```

```
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancerAttributes",
                "elasticloadbalancing:DescribeLoadBalancerPolicies",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DetachLoadBalancerFromSubnets",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:ModifyLoadBalancerAttributes",
                "elasticloadbalancing:ModifyTargetGroup",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets",
                "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
                "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:CreateServiceLinkedRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "kms:DescribeKey",
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

Below, you can find instructions on how to apply this policy.

Configuring IAM via the web interface

In order to configure IAM via the web interface, first create a new Policy and apply the previously created JSON file to it:

1. Open `Identity and Access Management (IAM)`.
2. Open the `Policies` page and click `Create Policy`.
3. Select the `JSON` tab and insert the policy.
4. Click `Next: Tags`, then `Next: Review`.
5. Enter a policy name in the `Name` field (e.g., `D8CloudProviderAWS`).
6. Click `Create Policy`.

Then add a new user:

1. Open the `Users` page of IAM and click `Add users`.
2. Enter a name in the `User name` field (e.g., `deckhouse`).

And apply the created Policy to it:

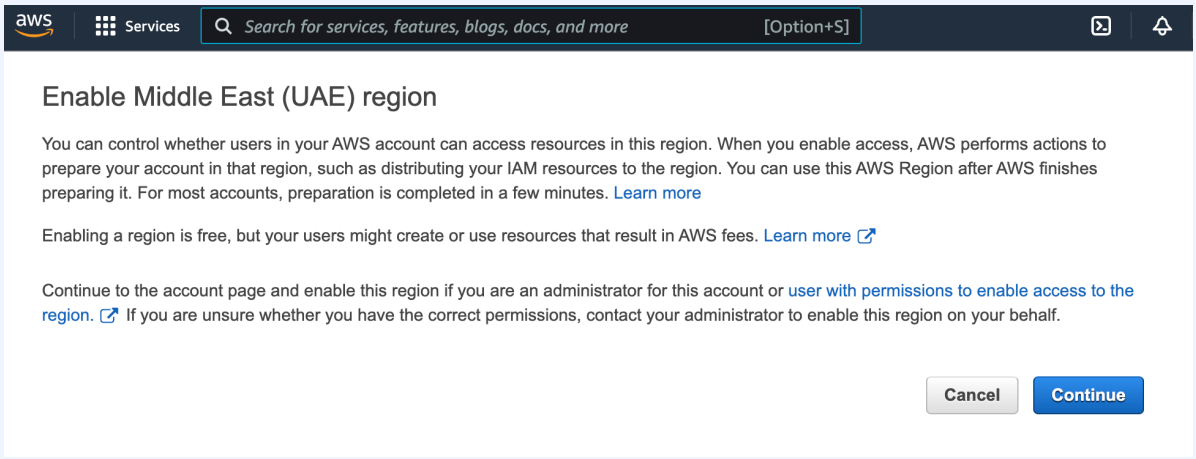
1. Select `Access key - Programmatic access` in the `Select AWS credential type` area.
2. Click `Next: Permissions`.
3. Select the `Attach existing policies directly` tab.
4. Search (use the `Filter policies` field) for the policy name entered above (e.g., `D8CloudProviderAWS`) and click the checkbox next to it.
5. Click `Next: Tags`, then `Next: Review`.
6. Click `Create user`.

Save credentials (`Access key ID` and `Secret access key`).

Make sure that you have access to the desired regions. Select the desired region from the dropdown list in the upper right corner.

If there is a switch to the selected region, then there is access to the region.

If there is no access to the region, then you will receive the following message (may differ):



In this case, click `Continue` to enable the region.

Configuring IAM via the CLI

Create the `JSON specification` using the following command.

```
cat > policy.json << EOF
<Policy JSON spec>
EOF
```

Create a new Policy based on the specification created above with `D8CloudProviderAWS` as a policy name and the ARN identifier:

```
aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
```

You will see the following:

```
{
    "Policy": {
        "PolicyName": "D8CloudProviderAWS",
        "PolicyId": "AAA",
        "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2020-08-27T02:52:06+00:00",
        "UpdateDate": "2020-08-27T02:52:06+00:00"
    }
}
```

Create a new user:

```
aws iam create-user --user-name deckhouse
```

You will see the following:

```
{
   "User": {
      "Path": "/",
      "UserName": "deckhouse",
      "UserId": "AAAXXX",
      "Arn": "arn:aws:iam::123:user/deckhouse",
      "CreateDate": "2020-08-27T03:05:42+00:00"
   }
}
```

You need to allow access to the API and remember your `AccessKeyId` + `SecretAccessKey` values:

```
aws iam create-access-key --user-name deckhouse
```

You will see the following:

```
{
   "AccessKey": {
      "UserName": "deckhouse",
      "AccessKeyId": "XXXYYY",
      "Status": "Active",
      "SecretAccessKey": "ZZZzzz",
      "CreateDate": "2020-08-27T03:06:22+00:00"
   }
}
```

Attach the specified `Policy` to the specified `User`:

```
aws iam attach-user-policy --user-name deckhouse --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
```

Configuring IAM via Terraform

An example of configuring IAM via Terraform:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name       = "D8CloudProviderAWS"
  path       = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON Policy>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user      = aws_iam_user.user.name
  policy_arn = aws_iam_policy.policy.arn
}
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> **i** The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

AWSInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of the AWS instance used by `machine-controller-manager` (the node-manager modile).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalSecurityGroups**
    array of strings

    The additional security groups to add to provisioned instances of the specific InstanceClass.

  - **spec.additionalTags**
    object

    Extra AWS tags for instances in addition to `AWSClusterConfiguration.tags`.

  - **spec.ami**
    string

    The Amazon Machine Image (AMI ID) to use in provisioned instances.

    If omitted, the AMI defined in the masterNodeGroup.instanceClass.ami parameter of the `AWSClusterConfiguration` resource is used.

    Here is how you can find the required AMI (each region has its own set of AMIs):

    ```
    aws ec2 --region <REGION> describe-images \
    --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
    jq '.Images[].ImageId'
    ```

    Example:

    ```
    ami: ami-040a1551f9c9d11ad
    ```

  - **spec.capacity**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

    Instance capacity for *non-standard* instance types (the `instanceTypes` parameter).

    Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

    The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`t2.micro`, `m6a.12xlarge`, etc.) is embedded in Deckhouse.

    - **spec.capacity.cpu**
      integer or string

      Required value

      Node vCPU resources.

      Pattern: `^[0-9]+m?$`

      Example:

```
cpu: 1000m
```

- **spec.capacity.memory**
  integer or string

  Required value

  Node memory resources.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

```
memory: 1500Mi
```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `20`

  Example:

```
diskSizeGb: 20
```

- **spec.diskType**
  string

  Instance EBS disk type.

  Default: `gp2`

  Example:

```
diskType: gp2
```

- **spec.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

```
instanceType: t3.large
```

- **spec.iops**
  string

  IOPS rate for io1 diskType

  Example:

```
iops: '500'
```

- **spec.spot**
  boolean

  Order preemptible virtual machines (Spot Instances).

  AWS can interrupt such a machine at any time.

  For more information about preemptible virtual machines, see the provider documentation.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of the AWS instance used by `machine-controller-manager` (the node-manager modile).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**

object

Required value

- **spec.additionalSecurityGroups**
  array of strings

  The additional security groups to add to provisioned instances of the specific InstanceClass.

- **spec.additionalTags**
  object

  Extra AWS tags for instances in addition to `AWSClusterConfiguration.tags`.

- **spec.ami**
  string

  The Amazon Machine Image (AMI ID) to use in provisioned instances.

  If omitted, the AMI defined in the masterNodeGroup.instanceClass.ami parameter of the `AWSClusterConfiguration` resource is used.

  Here is how you can find the required AMI (each region has its own set of AMIs):

  ```
  aws ec2 --region <REGION> describe-images \
  --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
  jq '.Images[].ImageId'
  ```

  Example:

  ```
  ami: ami-040a1551f9c9d11ad
  ```

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `instanceTypes` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`t2.micro`, `m6a.12xlarge`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `20`

Example:

```
diskSizeGb: 20
```

- **spec.diskType**
  string

  Instance EBS disk type.

  Default: `gp2`

  Example:

  ```
  diskType: gp2
  ```

- **spec.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

  ```
  instanceType: t3.large
  ```

- **spec.iops**
  string

  IOPS rate for io1 diskType

  Example:

  ```
  iops: '500'
  ```

- **spec.spot**
  boolean

  Order preemptible virtual machines (Spot Instances).

  AWS can interrupt such a machine at any time.

  For more information about preemptible virtual machines, see the provider documentation.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How do I create a peering connection between VPCs?

Let’s, for example, create a peering connection between two VPCs, vpc-a and vpc-b.

> **Caution!** IPv4 CIDR must be unique for each VPC.

To configure, follow these steps:

1. Switch to the region where vpc-a is running.

2. Click `VPC` -> `VPC Peering Connections` -> `Create Peering Connection` and then configure a peering connection:
   - Name: `vpc-a-vpc-b`.
   - Fill in `Local` and `Another VPC` fields.
3. Switch to the region where vpc-b is running.
4. Click `VPC` -> `VPC Peering Connections`.
5. Select the newly created peering connection and click `Action "Accept Request"`.
6. Add routes to vpc-b's CIDR over a peering connection to the vpc-a's routing tables.
7. Add routes to vpc-a's CIDR over a peering connection to the vpc-b's routing tables.

How do I create a cluster in a new VPC with access over an existing bastion host?

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Set up a peering connection using the instructions above.
3. Continue installing the cluster, enter `y` when asked about the Terraform cache:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

How do I create a cluster in a new VPC and set up bastion host to access the nodes?

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Manually set up the bastion host in the subnet -public-0.
3. Continue installing the cluster, enter `y` when asked about the Terraform cache:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Configuring a bastion host

There are two possible cases:

- A bastion host already exists in an external VPC; in this case, you need to:
  1. Create a basic infrastructure of the cluster: `dhctl bootstrap-phase base-infra`;
  2. Set up peering connection between an external and a newly created VPC;
  3. Continue the installation by specifying the bastion host: `dhctl bootstrap --ssh-bastion...`
- A bastion host needs to be deployed to a newly created VPC; in this case, you need to:
  1. Create a basic infrastructure of the cluster: `dhctl bootstrap-phase base-infra`;
  2. Manually run a bastion in the -public-0 subnet;
  3. Continue the installation by specifying the bastion host: `dhctl bootstrap --ssh-bastion...`

Adding CloudStatic nodes to a cluster

To add a pre-created VM as a node to a cluster, follow these steps:

1. Attach a security group `<prefix>-node` to the virtual machine.
2. Attach the IAM role `<prefix>-node` to the virtual machine.
3. Add the following tags to the virtual machine (so that `cloud-controller-manager` can find virtual machines in the cloud):

   ```
   "kubernetes.io/cluster/<cluster_uuid>" = "shared"
   "kubernetes.io/cluster/<prefix>" = "shared"
   ```

   - You can find out the `cluster_uuid` using the command:

     ```
     d8 k -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
     ```

   - You can find out `prefix` using the command:

     ```
     d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
     ```

     If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-
configuration.yaml"' \
  | base64 -d | grep prefix
```

How to increase the size of a volume?

Set the new size in the corresponding PersistentVolumeClaim resource, in the `spec.resources.requests.storage` parameter.

The operation is fully automatic and takes up to one minute. No further action is required.

The progress of the process can be observed in events using the command `d8 k describe pvc`.

> **i** After modifying a volume, you must wait at least six hours and ensure that the volume is in the `in-use` or `available` state before you can modify the same volume. This is sometimes referred to as a cooldown period. You can find details in the official documentation.

How do I configure access to Amazon ECR repository on cluster nodes?

1. Set permissions to read images in Repository policies. There must be an existing `Roles` object in `Principal`.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "RepositoryRead",
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::xxx:role/xxx-node"
         },
         "Action": [
           "ecr:BatchCheckLayerAvailability",
           "ecr:BatchGetImage",
           "ecr:DescribeImages",
           "ecr:DescribeRepositories",
           "ecr:GetAuthorizationToken",
           "ecr:GetDownloadUrlForLayer",
           "ecr:ListImages",
           "ecr:ListTagsForResource"
          ]
       }
     ]
   }
   ```

   Apply this policy in `Amazon ECR` > `Private registry` > `Repositories` > `` > `Permissions`.

2. Add `ecr:GetAuthorizationToken` to additionalRolePolicies.

How to create preemptible nodes in AWS?

Preemptible virtual machines in AWS are called Spot Instances. To make a node group order them, set the `spot` parameter of the AWSInstanceClass resource to `true`.

Example:

```
apiVersion: deckhouse.io/v1
kind: AWSInstanceClass
metadata:
  name: worker-spot
spec:
  instanceType: m5.large
  diskSizeGb: 30
  spot: true
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    minPerZone: 2
    maxPerZone: 4
    zones:
      - eu-central-1a
```

Handling AWS notifications

When an EC2 Instance Rebalance Recommendation signals an elevated risk of Spot Instance interruption, the `node-termination-handler` component automatically applies the `aws-node-termination-handler/rebalance-recommendation` taint with `effect: NoSchedule` to the CloudEphemeral node. This taint prevents the scheduler from placing new workloads on the node unless they have a matching toleration.

The `aws-node-termination-handler/spot-itn` taint indicates a notice of an upcoming Spot Instance interruption.

If either taint is present, the `node-manager` module cordons and drains the node, then deletes it. If the node count in a zone falls below the `minPerZone` parameter value, the group provisions new nodes.

The recommendation allows node decommissioning to start before an interruption notice arrives. AWS may send both notifications at the same time.

Limitations of preemptible nodes in AWS

Consider the following limitations:

- The `spot` parameter applies only to CloudEphemeral node groups, because AWSInstanceClass applies only to them.
- The time between the interruption notice and the instance shutdown may be too short to drain the node. Place only the workload that tolerates a sudden node loss on preemptible nodes.

> ⚠ Changing the `spot` parameter in an existing AWSInstanceClass causes all nodes of the corresponding group to be recreated.

How do I replace the OS image on the nodes?

The OS image is set by the `ami` parameter of the AWSInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the AWSClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

What to do if switching to nodes in lower-priority groups takes a long time?

If switching to nodes in lower-priority groups takes a long time (for example, when node groups with spot instances are set to the highest priority and, if such instances are unavailable, switching to nodes from other groups takes a very long time), follow the instructions.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

An example of the `AWSInstanceClass` custom resource

Below is a simple example of custom resource `AWSInstanceClass` configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSInstanceClass
metadata:
  name: worker
spec:
  instanceType: t3.large
  ami: ami-040a1551f9c9d11ad
  diskSizeGb: 15
  diskType:  gp2
```

LoadBalancer

Service object Annotations

The following parameters are supported in addition to the existing upstream ones:

1. `service.beta.kubernetes.io/aws-load-balancer-type` — if it has the `none` value, then the target group will **only** be created (without any LoadBalancer).
2. `service.beta.kubernetes.io/aws-load-balancer-backend-protocol` — this parameter is used together with `service.beta.kubernetes.io/aws-load-balancer-type: none`:
   - Possible values:
     - `tcp` (default);
     - `tls`;
     - `http`;
     - `https`.
   - **Caution!** The `cloud-controller-manager` (CCM) will try to recreate the target group in response to changes in this field. If the target group has NLB or ALB attached to it, the CCM will fail to delete it and get stuck in this state forever. You have to manually disconnect NLB or ALB from the target group.

Configuring the load balancer if Ingress nodes are not available in all zones

Set the following annotation for the Service object: `service.beta.kubernetes.io/aws-load-balancer-subnets: subnet-foo, subnet-bar`.

You can get current subnets for a particular installation as follows:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values cloud-provider-aws -o json \
| jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts

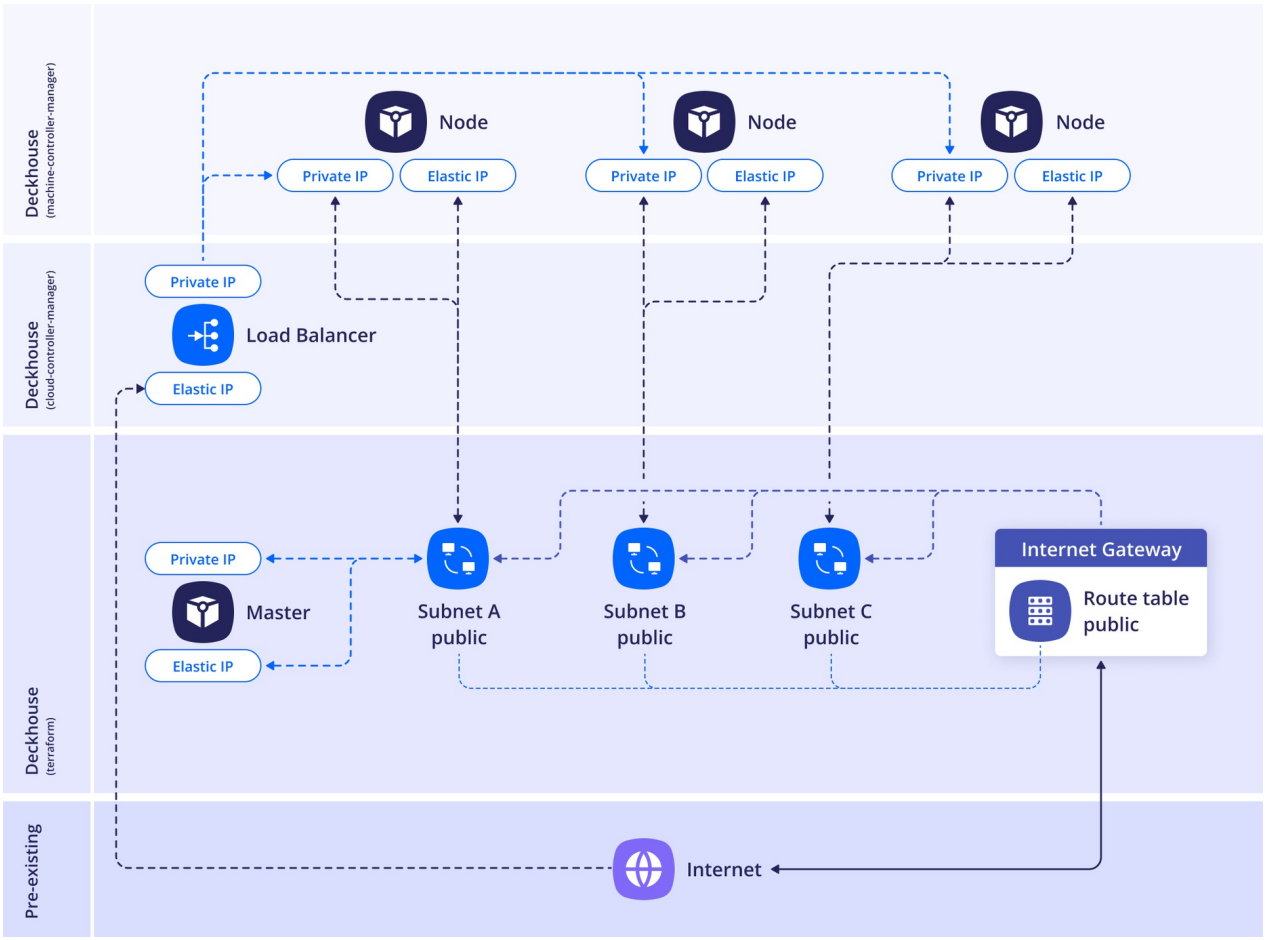
> The module lifecycle stage: General Availability

Two layouts are supported. Below is more information about each of them.

WithoutNAT

Recommended layout.

Under this placement strategy, each node gets a public IP (ElasticIP). NAT is not used at all.



When `disableDefaultSecurityGroup: false` is set, the module creates default security groups when a cluster is created.

The following groups and rules will be created:

- `<CLUSTER_PREFIX>-node` — assigned to cluster nodes:
  - allow any outgoing traffic to `0.0.0.0/0`
  - allow any incoming traffic from the `<CLUSTER_PREFIX>-loadbalancer` group
  - allow any incoming traffic from nodes in the same `<CLUSTER_PREFIX>-node` group
  - allow incoming traffic over the ICMP protocol from CIDRs listed in `publicNetworkAllowList` (default `0.0.0.0/0`)
- `<CLUSTER_PREFIX>-loadbalancer` — used by load balancers:
  - allow any incoming traffic from CIDRs in `publicNetworkAllowList`
  - allow any outgoing traffic to the `<CLUSTER_PREFIX>-node` group
- `<CLUSTER_PREFIX>-ssh-accessible` — created when `sshAllowList` is set; allows incoming traffic over the TCP protocol on port `22` from the listed CIDRs (default `0.0.0.0/0`). Assigned to master nodes or to the bastion host in the `WithNAT` layout

⚠ When `disableDefaultSecurityGroup: true` is set, you must create all required security groups yourself and specify them in `additionalSecurityGroups`. For load balancers, set groups with the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

Attach custom security groups (created in the cloud in advance) via `additionalSecurityGroups`:

- for master nodes — in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the AWSClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the AWSClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalSecurityGroups` parameter of the AWSInstanceClass resource

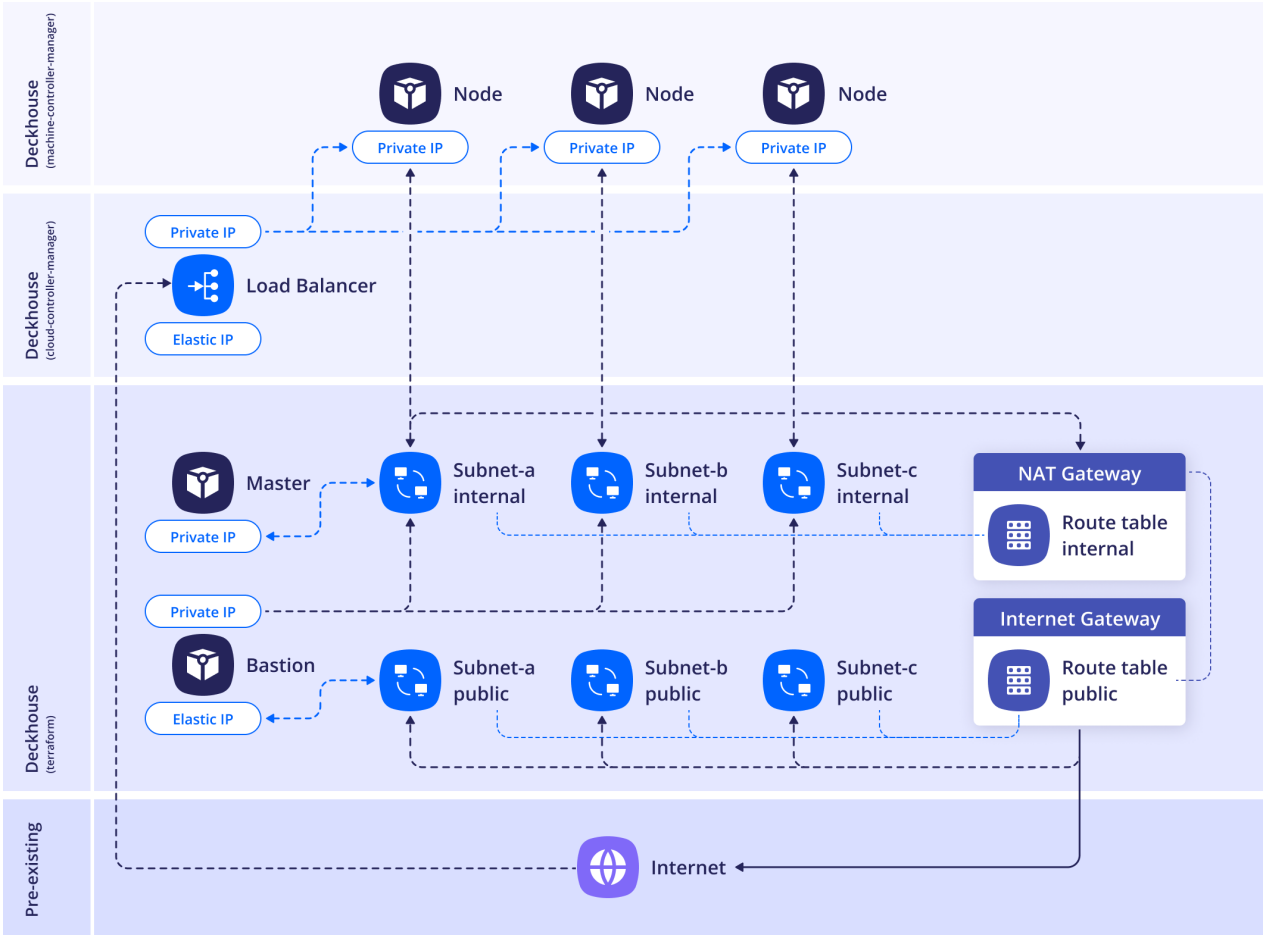
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  replicas: 1
  instanceClass:
    # Type of the instance.
    instanceType: m5.xlarge
    # Amazon Machine Image ID
    # AMI Catalog in the AWS console: EC2 -> AMI Catalog
    ami: ami-0aad10862ade98f27
    # Master node VM disk size.
    diskSizeGb: 30
    # Master node VM disk type to use.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: rangers
```

WithNAT

> **Caution!** A bastion host is required to access nodes (it can be created alongside the cluster by specifying the parameters in the section `withNAT.bastionInstance`).
>
> **Caution!** The NAT Gateway is always created in zone `a` in this layout. If cluster nodes are placed in other zones, then if there are problems in zone `a`, they will also be unavailable. In other words, when choosing the `WithNat` layout, the availability of the entire cluster will depend on the availability of zone `a`.

Virtual machines access the Internet using a NAT Gateway with a shared (and single) source IP.



When `disableDefaultSecurityGroup: false` is set, the module creates default security groups when a cluster is created.

The following groups and rules will be created:

- `<CLUSTER_PREFIX>-node` — assigned to cluster nodes:
  - allow any outgoing traffic to `0.0.0.0/0`
  - allow any incoming traffic from the `<CLUSTER_PREFIX>-loadbalancer` group
  - allow any incoming traffic from nodes in the same `<CLUSTER_PREFIX>-node` group
  - allow incoming traffic over the ICMP protocol from CIDRs listed in `publicNetworkAllowList` (default `0.0.0.0/0`)
- `<CLUSTER_PREFIX>-loadbalancer` — used by load balancers:
  - allow any incoming traffic from CIDRs in `publicNetworkAllowList`
  - allow any outgoing traffic to the `<CLUSTER_PREFIX>-node` group
- `<CLUSTER_PREFIX>-ssh-accessible` — created when `sshAllowList` is set; allows incoming traffic over the TCP protocol on port `22` from the listed CIDRs (default `0.0.0.0/0`). Assigned to master nodes or to the bastion host in the `WithNAT` layout

⚠ When `disableDefaultSecurityGroup: true` is set, you must create all required security groups yourself and specify them in `additionalSecurityGroups`. For load balancers, set groups with the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

Attach custom security groups (created in the cloud in advance) via `additionalSecurityGroups`:

- for master nodes — in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the AWSClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the AWSClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalSecurityGroups` parameter of the AWSInstanceClass resource

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Number of master nodes.
  # If there is more than one master node, the etcd cluster will be set up automatically.
  replicas: 1
  instanceClass:
    # Type of the instance.
    instanceType: m5.xlarge
    # Amazon Machine Image ID.
    # AMI Catalog in the AWS console: EC2 -> AMI Catalog
    ami: ami-0aad10862ade98f27
    # Master node VM disk size.
    diskSizeGb: 30
    # Master node VM disk type to use.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
tags:
  team: rangers
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — Azure

The module lifecycle stage: General Availability

The `cloud-provider-azure` module integrates Deckhouse Platform with Microsoft Azure. It allows the `node-manager` module to use Azure resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-azure` module:

- Managing Azure resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the Azure side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in Azure.
- Provisioning disks via the Azure Disk CSI driver (`disk.csi.azure.com`) and creating StorageClasses for Azure disk types so that PersistentVolumes can be requested from the cluster.

- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-azurerm`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the AzureInstanceClass resource.
- Registering with `node-manager` so that AzureInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

⚠ For utilities such as `ntpdate` and `chrony` to work correctly, make sure the load balancer has rules for UDP traffic. If outbound UDP is blocked, add a rule to the existing load balancer or create a Service of the LoadBalancer type with a UDP port.

Third-party components

List of third-party software used in the `cloud-provider-azure` module:

- Microsoft Azure Cloud provider v1.33.6, v1.34.3, v1.34.3, v1.36.1, v1.37.0

  License: Apache License 2.0

  Cloud Controller Manager for Microsoft Azure.

- Microsoft Azure CSI driver v1.33.8, v1.34.2, v1.34.2, v1.34.2, v1.36.0

  License: Apache License 2.0

  CSI driver for Microsoft Azure Disks.

- Microsoft Azure Terraform provider 3.117.1

  License: Mozilla Public License 2.0

  Terraform provider for Microsoft Azure Resource Manager.

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

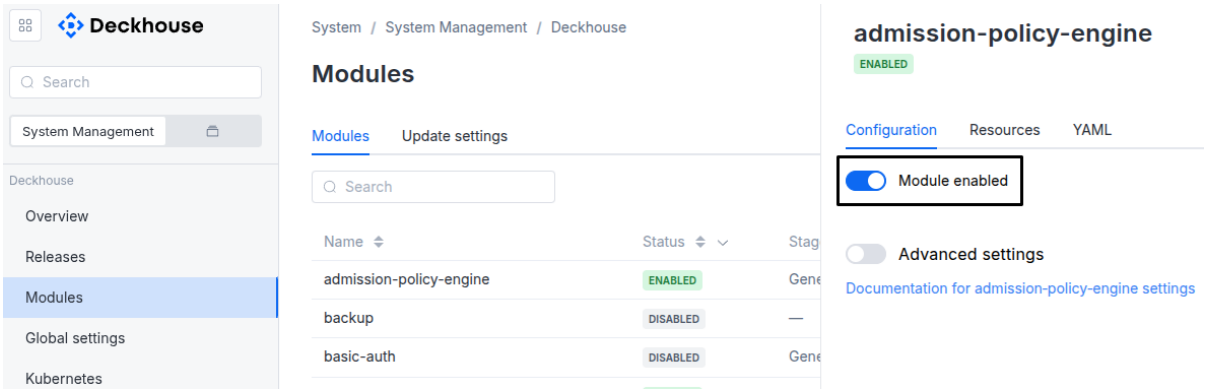
ℹ The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy (the AzureClusterConfiguration custom resource). In most cases, you do not need to configure the module manually.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-azure` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable cloud-provider-azure
```

- **Using ModuleConfig `cloud-provider-azure`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-azure` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-azure`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  enabled: true
```

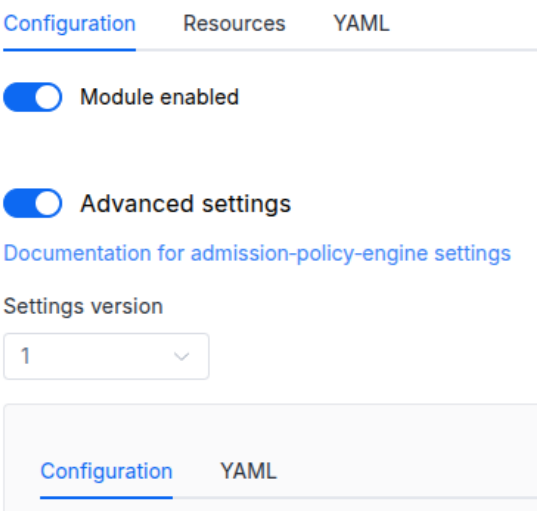
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-azure` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cloud-provider-azure` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-azure`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-azure` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-azure
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-azure` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `cloud-provider-azure`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the cloudInstances.ClassReference parameter). In the case of the Azure cloud provider, the instance class is the `AzureInstanceClass` custom resource that stores specific parameters of the machines.

The module automatically creates the following StorageClasses:

Name	Disk type
managed-standard-ssd	StandardSSD_LRS
managed-standard	Standard_LRS
managed-premium	Premium_LRS

It allows you to configure additional StorageClasses for volumes with configurable IOPS and Throughput. Also, you can filter out the unnecessary StorageClasses via the exclude parameter.

An example of Storage Class configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
      - name: managed-ultra-ssd
        diskIOPSReadWrite: 600
        diskMBpsReadWrite: 150
      exclude:
      - managed-standard.*
      - managed-premium
      default: managed-ultra-ssd
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - existing StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

```
default: pd-ssd-not-replicated
```

- **settings.storageClass.exclude**
  array of strings

  A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

  Example:

  ```
  exclude:
  - pd-standard.*
  - pd-ssd-replicated
  ```

- **settings.storageClass.provision**
  array of objects

  Defines additional StorageClasses for Azure ultra disks.

  - **settings.storageClass.provision.cachingMode**
    string

    String value that corresponds to the desired caching mode. If expected disk size is more than 4 TiB, you have to set `cachineMode: None`.

    Check out Azure documentation for more information.

    Allowed values: `None`, `ReadOnly`, `ReadWrite`

  - **settings.storageClass.provision.diskIOPSReadWrite**
    number

    Required value

    Disk IOPS (limit of 300 IOPS/GiB, up to a maximum of 160 K IOPS per disk).

  - **settings.storageClass.provision.diskMBpsReadWrite**
    number

    Required value

    Disk throughput in `MBps` (limit of a single disk is 256 KiB/s for each provisioned IOPS).

  - **settings.storageClass.provision.name**
    string

    Required value

    The name of the class to create.

  - **settings.storageClass.provision.tags**
    array of objects

    Azure disk tags.

    Example:

    ```
    tags:
    - key: key1
      value: val1
    - key: key2
      value: val2
    ```

    - **settings.storageClass.provision.tags.key**
      string

      Pattern: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

    - **settings.storageClass.provision.tags.value**
      string

      Pattern: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

  - **settings.storageClass.provision.type**
    string

    Required value

    Azure disk storage account type.

    Check out Azure documentation for more information.

Allowed values: `UltraSSD_LRS`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

AzureClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Azure.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
vNetCIDR: 10.0.0.0/16
nameservers:
  addresses:
  - 8.8.8.8
  - 8.8.4.4
subnetCIDR: 10.0.0.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202207120
    enableExternalIP: false
provider:
  subscriptionId: "<SUBSCRIPTION_ID>"
  location: westeurope
  clientId: "<CLIENT_ID>"
  clientSecret: "<CLIENT_SECRET>"
  tenantId: "<TENANT_ID>"
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**
  string

  Required value

  Allowed values: `AzureClusterConfiguration`

- **layout**

string

Required value

The way resources are located in the cloud.

Read more about possible provider layouts.

Allowed values: `Standard`

- **masterNodeGroup**

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the AzureInstanceClass fields.

    - **masterNodeGroup.instanceClass.acceleratedNetworking**
      boolean

      Accelerated Networking provides up to 30Gbps in networking throughput.

      Default: `false`

    - **masterNodeGroup.instanceClass.additionalTags**
      object

      The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **masterNodeGroup.instanceClass.diskSizeGb**
      integer

      Instance root disk size in gibibytes.

      Example:

      ```
      diskSizeGb: 40
      ```

    - **masterNodeGroup.instanceClass.diskType**
      string

      The type of the volumes to create.

      You can view a list of available volume types using the Azure CLI:

      ```
      az vm list-skus -l westeurope --zone
      ```

      Example:

      ```
      diskType: StandardSSD_LRS
      ```

    - **masterNodeGroup.instanceClass.enableExternalIP**
      boolean

      Defines whether to enable external IP for an instance or not.

      Only available for the `Standard` layout.

      Default: `false`

      Allowed values: `true`, `false`

    - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
      integer

Etcd disk size in gibibytes.

Default: `20`

Example:

```
etcdDiskSizeGb: 20
```

- **masterNodeGroup.instanceClass.machineSize**
  string

  Required value

  The type of instances to provision.

  Getting a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **masterNodeGroup.instanceClass.urn**
  string

  Required value

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official Azure documentation.

  Getting the `urn` via Azure CLI (column #4):

  ```
  az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o
  table
  az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
  ```

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version you use).

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A list of zones where master nodes can be created.

  You can browse a list of zones available for the selected instance type using the Azure CLI:

  ```
  az vm list-skus -l westeurope -o table
  ```

  Default: `[1,2,3]`

- **nameservers**
  object

  DNS server configuration for the Azure virtual network used by the cluster nodes.

  - **nameservers.addresses**
    array of strings

    A list of DNS servers.

    Example:

```
addresses:
- 8.8.8.8
- 8.8.4.4
```

- **Element of the array**
  string

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the AzureInstanceClass.

    - **nodeGroups.instanceClass.acceleratedNetworking**
      boolean

      Accelerated Networking provides up to 30Gbps in networking throughput.

      Default: `false`

    - **nodeGroups.instanceClass.additionalTags**
      object

      The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **nodeGroups.instanceClass.diskSizeGb**
      integer

      Instance root disk size in gibibytes.

      Example:

      ```
      diskSizeGb: 40
      ```

    - **nodeGroups.instanceClass.diskType**
      string

      The type of the volumes to create.

      You can view a list of available volume types using the Azure CLI:

      ```
      az vm list-skus -l westeurope --zone
      ```

      Example:

      ```
      diskType: StandardSSD_LRS
      ```

    - **nodeGroups.instanceClass.enableExternalIP**
      boolean

      Defines whether to enable external IP for an instance or not.

      Only available for the `Standard` layout.

      Default: `false`

      Allowed values: `true`, `false`

    - **nodeGroups.instanceClass.machineSize**
      string

      Required value

      The type of instances to provision.

Getting a list of available types for the specific region using the Azure CLI:

```
az vm list-sizes --location westeurope -o table
```

Example:

```
machineSize: Standard_F4
```

- **nodeGroups.instanceClass.urn**
  string

  Required value

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official Azure documentation.

  Getting the `urn` via Azure CLI (column #4):

  ```
  az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o
  table
  az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
  ```

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version you use).

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Format — `key: value`.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

        - **nodeGroups.nodeTemplate.taints.effect**
          string

          Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
        - **nodeGroups.nodeTemplate.taints.key**
          string
        - **nodeGroups.nodeTemplate.taints.value**
          string
    - **nodeGroups.replicas**
      integer

      Required value

      The number of nodes to create.
    - **nodeGroups.zones**
      array of strings

      A list of zones where static nodes can be created;

      You can browse a list of zones available for the selected instance type using the Azure CLI:

      ```
      az vm list-skus -l westeurope -o table
      ```

      Default: `[1,2,3]`
- **peeredVNets**
  array of objects

  An array of `VNets` to merge with the cluster network.

  The service account must have access to all the `VNets` listed above. You have to configure the peering connection manually if no access is available.

  - **peeredVNets.resourceGroupName**
    string

    Required value

    The name of the resource group with the VNet.
  - **peeredVNets.vnetName**
    string

    Required value

    The name of the VNet.
- **provider**
  object

  Required value

  Parameters for connecting to the Azure API.

  - **provider.clientId**
    string

    Required value

    The client ID.
  - **provider.clientSecret**
    string

    Required value

    The client’s secret.

    Keep in mind the expiration date of the secret. By default, it is valid for one year. Refer to the official documentation to create a service account with a longer secret expiration date.
  - **provider.location**
    string

    Required value

The name of the geo location to create all the resources. Getting available locations from Azure CLI:

```
az account list-locations -o table
```

- **provider.subscriptionId**
  string

  Required value

  The ID of the subscription.

- **provider.tenantId**
  string

  Required value

  The ID of the tenant.

- **serviceEndpoints**
  array of strings

  The list of Service endpoints to associate with the subnet.

  Virtual Network (VNet) service endpoint provides secure and direct connectivity to Azure services over an optimized route over the Azure backbone network. Endpoints allow you to secure your critical Azure service resources to only your virtual networks. Service Endpoints enables private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

  More information about Service Endpoints can be found in the official documentation.

  - **Element of the array**
    string

    Allowed values: `Microsoft.AzureActiveDirectory`, `Microsoft.AzureCosmosDB`, `Microsoft.ContainerRegistry`, `Microsoft.CognitiveServices`, `Microsoft.EventHub`, `Microsoft.KeyVault`, `Microsoft.ServiceBus`, `Microsoft.Sql`, `Microsoft.Storage`, `Microsoft.Storage.Global`, `Microsoft.Web`

- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, `*`.

- **sshPublicKey**
  string

  Required value

  Public key to access nodes as `azureuser`.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.natGatewayPublicIpCount**
    integer

    The number of IP addresses for the NAT Gateway (pricing). `NAT Gateway` is not used if the value is `0`.

    Default: `0`

- **subnetCIDR**
  string

  Required value

  A subnet from the `vNetCIDR` address space for cluster nodes.

  A virtual network can be segmented into one or more subnets up to the limits.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  Example:

  ```
  subnetCIDR: 10.1.2.0/24
  ```

- **tags**
  object

  A list of tags in the `key: value` format to attach to all cluster resources.

You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **vNetCIDR**
  string

  Required value

  An address space of the virtual network in the CIDR format.

  A virtual network is a virtual, isolated portion of the Azure public network. Each virtual network is dedicated to your subscription.

  Warning. If you are setting up peering, using vpn or linking networks of other clusters, network address spaces should not overlap.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  Example:

  ```
  vNetCIDR: 10.0.0.0/16
  ```

- **zones**
  array of strings

  The globally restricted set of zones that this Cloud Provider works with.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

The module lifecycle stage: General Availability

To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

Only regions where `Availability Zones` are available are supported.

The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

⚠ Starting with DP version 1.77, Azure uses the `cilium` CNI by default for new clusters. Existing clusters keep the current CNI configuration.

New clusters require Linux kernel version 5.8 or newer on all nodes. Make sure firewalls or security groups allow inter-node UDP traffic for Cilium VXLAN. For more information about Cilium requirements, limitations, and behavior, see the installation requirements, Network interaction of the platform components, and the `cni-cilium` module documentation.

To rule the Microsoft Azure cloud, you need an account and at least a single Subscription connected to id.

You have to create a service account with Microsoft Azure so that Deckhouse Platform can manage cloud resources:

1. Install the Azure CLI, login and get Subscription ID:

   ```
   export SUBSCRIPTION_ID=$(az login | jq -r '.[0].id')
   ```

2. Create the service account:

   ```
   az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --
   name "DeckhouseCANDI"
   ```

   Example output of the command:

   ```
   {
     "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx",     <-- used in the clientId
   parameter of the AzureClusterConfiguration resource
     "displayName": "DeckhouseCANDI",
     "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", <-- used in the clientSecret
   parameter of the AzureClusterConfiguration resource
     "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"    <-- used in the tenantId
   parameter of the AzureClusterConfiguration resource
   }
   ```

   ℹ By default, service account will be created with a secret (used in the clientSecret parameter of the `AzureClusterConfiguration` resource) validity period of one year without automatic renewal. Refer to the official documentation to create a service account with a longer secret expiration date.

You have to be logged in for further work with the `az` tool. Use the service account username, password, and tenant to log in:

```
az login --service-principal -u <username> -p <password> --tenant <tenant>
```

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

ℹ The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

ℹ The module lifecycle stage: General Availability

AzureInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of the Azure Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.acceleratedNetworking**
    boolean

    Accelerated Networking provides up to 30Gbps in networking throughput.

    Default: `true`

  - **spec.additionalTags**
    object

    The additional tags to attach to the instances created.

  - **spec.capacity**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

    Instance capacity for *non-standard* instance types (the `machineSize` parameter).

    Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

    The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`Standard_A1_v2`, `Standard_B16ms`, etc.) is embedded in Deckhouse.

    - **spec.capacity.cpu**
      integer or string

      Required value

      Node vCPU resources.

      Pattern: `^[0-9]+m?$`

      Example:

      ```
      cpu: 1000m
      ```

    - **spec.capacity.memory**
      integer or string

      Required value

      Node memory resources.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

      Example:

      ```
      memory: 1500Mi
      ```

  - **spec.diskSizeGb**
    integer

    Instance root disk size in gibibytes.

    Default: `50`

    Example:

    ```
    diskSizeGb: 40
    ```

  - **spec.diskType**
    string

    The type of the volume to create.

    You can view a list of available volume types using the Azure CLI:

    ```
    az vm list-skus -l westeurope --zone`
    ```

Default: `StandardSSD_LRS`

- **spec.machineSize**
  string

  Required value

  The type of instances to provision.

  You can view a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **spec.urn**
  string

  Not required value.

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official documentation.

  You can get the latest version of an image using the Azure CLI:

  - `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope`;
  - `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope`.

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  Example:

  ```
  urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of the Azure Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.acceleratedNetworking**
    boolean

    Accelerated Networking provides up to 30Gbps in networking throughput.

    Default: `true`

  - **spec.additionalTags**
    object

    The additional tags to attach to the instances created.

  - **spec.capacity**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

    Instance capacity for *non-standard* instance types (the `machineSize` parameter).

    Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`Standard_A1_v2`, `Standard_B16ms`, etc.) is embedded in Deckhouse.

- **spec.capacity.cpu**
  integer or string

  Required value

  Node vCPU resources.

  Pattern: `^[0-9]+m?$`

  Example:

  ```
  cpu: 1000m
  ```

- **spec.capacity.memory**
  integer or string

  Required value

  Node memory resources.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  memory: 1500Mi
  ```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 40
  ```

- **spec.diskType**
  string

  The type of the volume to create.

  You can view a list of available volume types using the Azure CLI:

  ```
  az vm list-skus -l westeurope --zone`
  ```

  Default: `StandardSSD_LRS`

- **spec.machineSize**
  string

  Required value

  The type of instances to provision.

  You can view a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **spec.urn**
  string

  Not required value.

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official documentation.

  You can get the latest version of an image using the Azure CLI:

  - `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope`;

- `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope`.

By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

Example:

```
urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

An example of the `AzureInstanceClass` custom resource

Below is a simple example of the `AzureInstanceClass` custom resource:

```
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: example
spec:
  machineSize: Standard_F4
```

Replacing the OS image

In this example, the nodes of a group are moved to a new OS image. The new image is set in the `urn` parameter of the instance class that the node group refers to:

```
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: example
spec:
  machineSize: Standard_F4
  urn: Canonical:ubuntu-24_04-lts:server:24.04.202602040
```

After the instance class is changed, Deckhouse Platform recreates the `CloudEphemeral` nodes from the new image. For `CloudPermanent` nodes and master nodes, the image is set in the `instanceClass` section of the AzureClusterConfiguration resource. The replacement procedure is described in “How do I replace the OS image in a cloud cluster?”.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts

> i The module lifecycle stage: General Availability

> i **Caution!** Only regions where `Availability Zones` are available are supported.

Standard

- A separate resource group is created for the cluster.
- By default, one external IP address is dynamically allocated to each instance (it is used for Internet access only). Each IP has 64000 ports available for SNAT.
- The NAT Gateway (pricing) is supported. With it, you can use static public IP addresses for SNAT.
- Public IP addresses can be assigned to master nodes and nodes created by Terraform.
- If the master does not have a public IP, then an additional instance with a public IP (aka bastion host) is required for installation tasks and access to the cluster. In this case, you will also need to configure peering between the cluster's VNet and bastion's VNet.
- Peering can also be configured between the cluster VNet and other VNets.

When a cluster is created, the module creates a Network Security Group (NSG) named after the cluster prefix and associates it with the node subnet.

The following rules will be created:

- `AllowIcmp` — allow incoming traffic over the ICMP protocol from any source
- `AllowSsh` — allow incoming traffic over the TCP protocol on port `22` from CIDRs listed in `sshAllowList` (from any source if the list is not set)

You cannot attach a pre-created custom NSG to nodes through module parameters. Use `sshAllowList` to restrict SSH access.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>" # Required.
vNetCIDR: 10.50.0.0/16 # Required.
subnetCIDR: 10.50.0.0/24 # Required.
standard:
  natGatewayPublicIpCount: 1 # Optional, by default 0.
masterNodeGroup:
  replicas: 1
  zones: ["1"] # Optional, by default ["1"].
  instanceClass:
    machineSize: Standard_F4 # Required.
    diskSizeGb: 32
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140 # Required.
    enableExternalIP: false # Optional, by default true.
provider:
  subscriptionId: "<SUBSCRIPTION_ID>" # Required.
  clientId: "<CLIENT_ID>" # Required.
  clientSecret: "<CLIENT_SECRET>" # Required.
  tenantId: "<TENANT_ID>" # Required.
  location: "westeurope" # Required.
# Optional, list of Azure VNets with which Kubernetes VNet will be peered.
peeredVNets:
  - resourceGroupName: kube-bastion # Required.
    vnetName: kube-bastion-vnet # Required.
```

> i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — Basis Dynamix

> **i** Included in extensions: **On-prem IaaS**

> ⚠ The module lifecycle stage: Preview

The `cloud-provider-dynamix` module integrates Deckhouse Platform with the Basis Dynamix platform. It allows the `node-manager` module to use Dynamix resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-dynamix` module:

- Managing Dynamix resources via `cloud-controller-manager`:
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in Dynamix;
  - creates load balancers for Services of the LoadBalancer type. The Service must include annotations with the names of the internal and external networks.
- Provisioning disks via the Dynamix CSI driver (`dynamix.deckhouse.io`) so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-decort`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the DynamixInstanceClass resource.
- Registering with `node-manager` so that DynamixInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> **i** The module is in the Preview stage.

Third-party components

List of third-party software used in the `cloud-provider-dynamix` module:

- Dynamix Terraform provider 4.12.2

  License: Apache License 2.0

  Terraform provider for Dynamix (DECORT).

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** Included in extensions: **On-prem IaaS**

> ⚠ The module lifecycle stage: Preview

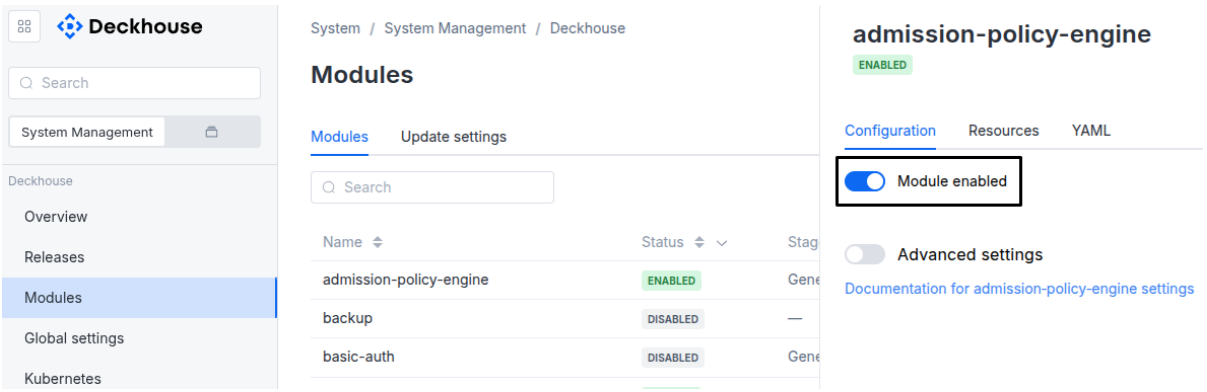
The module is automatically enabled for all cloud clusters deployed in Basis Dynamix.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dynamix` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-dynamix
  ```

- **Using ModuleConfig `cloud-provider-dynamix`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-dynamix` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-dynamix`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-dynamix
  spec:
    enabled: true
  ```

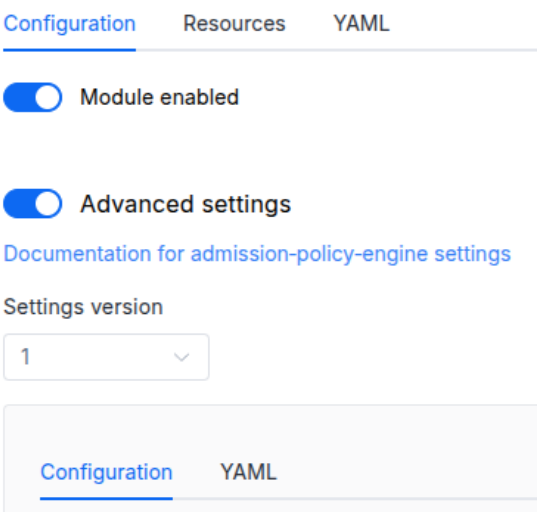
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dynamix` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-dynamix` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-dynamix`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `cloud-provider-dynamix` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-dynamix
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-dynamix` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-dynamix`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dynamix
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object

    The module automatically creates a StorageClass for every `ENABLED` storage policy available to the cluster account.

    - **settings.storageClass.default**
      Deprecated
      string

      The name of the StorageClass that the module marks as the default one among the StorageClasses it creates.

      The parameter applies only if the cluster has no default StorageClass and `global.defaultClusterStorageClass` isn’t set.

      If the parameter is omitted, the default StorageClass is either:

      - an arbitrary StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: storage-policy01
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

      Example:

      ```
      exclude:
      - ".*-hdd"
      - storage-policy01
      ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> ⚠ The module lifecycle stage: Preview

> If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-dynamix` module in the Deckhouse configuration. If the cluster control plane is hosted in a cloud, the cloud provider uses the DynamixClusterConfiguration structure for configuration.

DynamixClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Dynamix.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: ssh-rsa AAAA
location: dynamix
account: acc_user
storagePolicy: storage_policy01
nodeNetworkCIDR: 10.241.32.0/24
nameservers:
- 10.0.0.10
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

- **account**
  string

  Required value

  Account name.

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **kind**
  string

  Required value

  Allowed values: `DynamixClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **location**
  string

  Required value

  The name of the Basis Dynamix location where the cluster resource group is created.

- **masterNodeGroup**
  object

  Required value

The definition of the master’s NodeGroup.

> **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

- **masterNodeGroup.instanceClass**
  object

  Required value

  Partial contents of the fields of the DynamixInstanceClass.

  - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
    integer

    Etcd disk size in GiB.

    Default: `15`

    Example:

    ```
    etcdDiskSizeGb: 15
    ```

  - **masterNodeGroup.instanceClass.externalNetwork**
    string

    Required value

    External network name.

    Example:

    ```
    externalNetwork: extnet_vlan_1700
    ```

  - **masterNodeGroup.instanceClass.imageName**
    string

    Required value

    OS image name to use for VM deployment.

    Example:

    ```
    imageName: '1'
    ```

  - **masterNodeGroup.instanceClass.memory**
    integer

    Required value

    Memory in MiB to allocate to the VM.

    Allowed values: `1 <= X`

    Example:

    ```
    memory: 8192
    ```

  - **masterNodeGroup.instanceClass.numCPUs**
    integer

    Required value

    Number of vCPUs to allocate to the VM.

    Allowed values: `1 <= X`

    Example:

    ```
    numCPUs: 2
    ```

  - **masterNodeGroup.instanceClass.rootDiskSizeGb**
    integer

    Root disk size in GiB to use in Dynamix VirtualMachines.

    Default: `50`

    Example:

```
rootDiskSizeGb: 50
```

- **masterNodeGroup.instanceClass.storagePolicy**
  string

  The name of the storage policy for this node group instead of the cluster-wide `storagePolicy`.

  > **Warning.** A new value applies only to the nodes created after the change. Existing nodes aren’t recreated.

  Example:

  ```
  storagePolicy: storage_policy01
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **nameservers**
  array of strings

  A list of IPv4 addresses of the DNS servers for the nodes.

  - In the `StandardWithInternalNetwork` layout, the nodes get these addresses through the ViNS.
  - In the `Standard` layout, the addresses are written into the cloud-init of every node. Specify them if the external network doesn’t provide DNS server addresses to the nodes. Otherwise, the nodes can’t resolve names.

  A new value applies only to the nodes created after the change.

  Default: `[]`

  - **Element of the array**
    string

    Pattern: `^[0-9.]+$`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the.

    - **nodeGroups.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in GiB.

      Default: `15`

      Example:

      ```
      etcdDiskSizeGb: 15
      ```

    - **nodeGroups.instanceClass.externalNetwork**
      string

      Required value

      External network name.

      Example:

      ```
      externalNetwork: extnet_vlan_1700
      ```

    - **nodeGroups.instanceClass.imageName**
      string

Required value

OS image name to use for VM deployment.

Example:

```
imageName: '1'
```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.numCPUs**
  integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **nodeGroups.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in Dynamix VirtualMachines.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **nodeGroups.instanceClass.storagePolicy**
  string

  The name of the storage policy for this node group instead of the cluster-wide `storagePolicy`.

  > **Warning.** A new value applies only to the nodes created after the change. Existing nodes aren’t recreated.

  Example:

  ```
  storagePolicy: storage_policy01
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

        - **nodeGroups.nodeTemplate.labels**
          object

          A list of labels to attach to cluster resources.

          The same as the `metadata.labels` standard field.

          Example:

          ```
          labels:
            environment: production
            app: warp-drive-ai
          ```

        - **nodeGroups.nodeTemplate.taints**
          array of objects

          The same as the `.spec.taints` field of the Node object.

          > **i** Available fields: `effect`, `key`, and `values`.

          Example:

          ```
          taints:
          - effect: NoExecute
            key: ship-class
            value: frigate
          ```

          - **nodeGroups.nodeTemplate.taints.effect**
            string

            Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
          - **nodeGroups.nodeTemplate.taints.key**
            string
          - **nodeGroups.nodeTemplate.taints.value**
            string
    - **nodeGroups.replicas**
      integer

      Required value

      The number of nodes to create.

- **nodeNetworkCIDR**
  string

  IP CIDR to be assigned to the internal segment of this ViNS.

- **provider**
  object

  Required value

  Parameters for connecting to the Dynamix.

    - **provider.appId**
      string

      Required value

      App ID.

    - **provider.appSecret**
      string

      Required value

      App Secret.

    - **provider.controllerUrl**
      string

      Required value

      The URL to the Dynamix API endpoint.

    - **provider.insecure**
      boolean

      Set to `true` if Dynamix has a self-signed certificate.

Default: `false`

- **provider.oAuth2Url**
  string

  Required value

  The URL to the SSO Dynamix API endpoint.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **storagePolicy**
  string

  Required value

  The name of the storage policy used by default for virtual machine disks.

  A storage policy defines the storage endpoints and pools available to the account, as well as an IOPS limit. Basis Dynamix chooses the exact placement of a disk within the policy. You can set a different storage policy for a node group in its instance class.

  > **Warning.** Changing the value recreates the nodes of the `CloudEphemeral` type in all node groups, including groups with their own storage policy. Master nodes and the nodes of `nodeGroups` aren't recreated.

  Example:

  ```
  storagePolicy: storage_policy01
  ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> ⚠ The module lifecycle stage: Preview

Platform version

The module requires Basis Dynamix 4.6 or newer. Starting with version 4.6, Basis Dynamix creates disks and virtual machines only with a specified storage policy.

Before creating any resources in the cloud, `dhctl bootstrap` runs the `dynamix-storage-policy` preflight check. On Basis Dynamix earlier than 4.6, this check stops the installation.

Account permissions

The module works with Basis Dynamix through the user API (`/restmachine/cloudapi/*`). Use an account without platform administrator permissions (`/restmachine/cloudbroker/*`) for the module. With administrator permissions, the module gets access to the resources of other accounts, which breaks account isolation.

The account specified in the `provider` section of DynamixClusterConfiguration must be allowed to use the following user API groups:

Group	Component	Purpose
`account`, `locations`, `rg`	All components	Finding the account, the location, and the resource group by name
`extnet`, `vins`	Terraform, CAPD	External and internal networks
`image`	Terraform, CAPD	OS images for the nodes

Group	Component	Purpose
`storage_policy`	Terraform, CAPD, CSI, cloud-data-discoverer	Listing the storage policies available to the account, finding a policy by name, attaching policies to the cluster resource group
`disks`	Terraform, CSI	Disks of master nodes and PersistentVolumes
`compute`, `kvmx86`	Terraform, CCM, CAPD, CSI	Virtual machines and disk attachment
`lb`	CCM	Load balancers for Services of the LoadBalancer type

Storage policies in the cluster resource group

Basis Dynamix creates a virtual machine only if its storage policy is attached to the resource group. Therefore, when the module creates the cluster resource group, it attaches all `ENABLED` storage policies of the account to it.

The list of attached storage policies is updated when you run `dhctl converge` and when the `terraform-manager` module applies changes automatically. During the update, the module:

- Attaches the storage policies that have become available to the account.
- Detaches the storage policies that have become unavailable to the account or whose status has changed from `ENABLED` to another one.

Disable a storage policy only when the cluster nodes and volumes no longer use it. When a storage policy becomes unavailable to the account, the module deletes its StorageClass from the cluster. The volumes created in this policy are kept.

> ⚠ When the cluster is deleted or its installation is aborted, the module force-deletes the cluster resource group, bypassing the recycle bin. Everything still inside the group is deleted permanently with it, including volume disks, load balancers, and the virtual machines of node groups. The etcd disks of master nodes and the ViNS are also deleted bypassing the recycle bin. Move the data you need out of the cluster before deleting it.

Prepare an operating system image

Operating system vendors typically provide special cloud builds of their operating systems for use in virtualization environments. These builds typically contain virtual hardware drivers, cloud-init, virtualization guest agents, and are distributed as IMG or QCOW2 disk images. We recommend that you use these cloud images as the OS on the nodes in your clusters.

The cloud image of the operating system must be placed in the “Images” → “Template Images” section of the Basis Dynamix portal. Follow these steps to upload the OS image to the storage:

If the infrastructure does not have a DNS server, access to the Basis Dynamix portal from the cluster. Basis Dynamix portal from the cluster can be organized by adding the IP address and domains associated with the Basis Dynamix portal to the `cloud-init` template for generating the hosts file. This template is located in the `/etc/cloud/templates/` folder. The name of the template depends on the OS.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> ⚠ After adding data to hosts and before creating the template from the virtual machine, you must run the `cloud-init clean` command.

Replacing the OS image

The OS image is set by the `imageName` parameter of the DynamixInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the DynamixClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> ⚠ The module lifecycle stage: Preview

DynamixInstanceClass

Scope: Cluster
Version: v1

Parameters of a group of Dynamix VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.externalNetwork**
    string

    External network name.

    Example:

    ```
    externalNetwork: extnet_vlan_1700
    ```

  - **spec.imageName**
    string

    Required value

    Name of the OS image to base this VM on.

    Example:

    ```
    imageName: alt-p10-cloud-x86_64.img
    ```

  - **spec.memory**
    integer

    Required value

    Memory in MiB to allocate to Dynamix VirtualMachines.

    Example:

    ```
    memory: 8192
    ```

  - **spec.numCPUs**
    integer

    Required value

    Count of vCPUs to allocate to Dynamix VirtualMachines.

    Example:

    ```
    numCPUs: 2
    ```

  - **spec.rootDiskSizeGb**
    integer

Root disk size in GiB to use in Dynamix VirtualMachines.

The disk will be automatically resized if its size in the template differs from specified.

Default: `30`

Example:

```
rootDiskSizeGb: 30
```

- **spec.storagePolicy**
  string

  The name of the storage policy for the virtual machines of this instance class instead of the cluster-wide `storagePolicy`.

  > **Warning.** Changing the value recreates the nodes of the node groups that use this instance class.

  Example:

  ```
  storagePolicy: storage_policy01
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> ⚠ The module lifecycle stage: Preview

How to configure a LoadBalancer?

To configure a Service of the LoadBalancer type, add the following annotations to the Service manifest:

```
metadata:
  annotations:
    dynamix.cpi.flant.com/internal-network-name: <internal_name>
    dynamix.cpi.flant.com/external-network-name: <external_name>
```

Both annotations are required:

- `dynamix.cpi.flant.com/internal-network-name` — the name of the internal network in Basis Dynamix
- `dynamix.cpi.flant.com/external-network-name` — the name of the external network in Basis Dynamix

The terms “internal network” and “external network” are used in the context of Basis Dynamix. The external network does not have to be public and may use private IP addresses.

If one of the annotations is not specified, cloud-controller-manager will fail to process the Service.

Other ways to handle inbound traffic to the cluster, such as an external load balancer or MetalLB, are described in the Load balancing section.

How is the disk placement chosen?

The disk placement is defined by the storage policy. Its name is set in the following parameters:

- `storagePolicy` at the root of DynamixClusterConfiguration: The storage policy for all virtual machines in the cluster.
- `storagePolicy` in `masterNodeGroup.instanceClass`, `nodeGroups[].instanceClass`, or DynamixInstanceClass: The storage policy for a single node group instead of the cluster-wide one.

A storage policy sets the storage endpoint and pool pairs available to the account, as well as an IOPS limit. Basis Dynamix chooses the exact placement of a disk within the policy.

Which nodes are recreated when the storage policy changes?

The storage policy of a disk is set when the virtual machine is created and doesn’t change for an existing machine. Therefore, when the storage policy changes, the module recreates nodes of the `CloudEphemeral` type:

- When `storagePolicy` changes in a DynamixInstanceClass: The nodes of the groups that use this DynamixInstanceClass.
- When `storagePolicy` changes at the root of DynamixClusterConfiguration: The nodes of all groups, including groups with their own storage policy.

The module doesn’t recreate master nodes and the nodes of the `nodeGroups` groups. A new storage policy applies to such a node only when the node is created again.

Which StorageClasses does the module create?

The module creates a StorageClass for every storage policy that is available to the cluster account and has the `ENABLED` status.

The module derives the StorageClass name from the storage policy name:

- Converts letters to lowercase
- Replaces spaces with `-`
- Removes the characters that a Kubernetes object name can’t contain, including `_`

For example, for the `storage_policy01` storage policy, the module creates the `storagepolicy01` StorageClass. If the names of several storage policies result in the same StorageClass name, the module creates the StorageClass for the policy whose name comes first when sorted.

To prevent the module from creating some StorageClasses, list their names or regular expressions in the `storageClass.exclude` parameter.

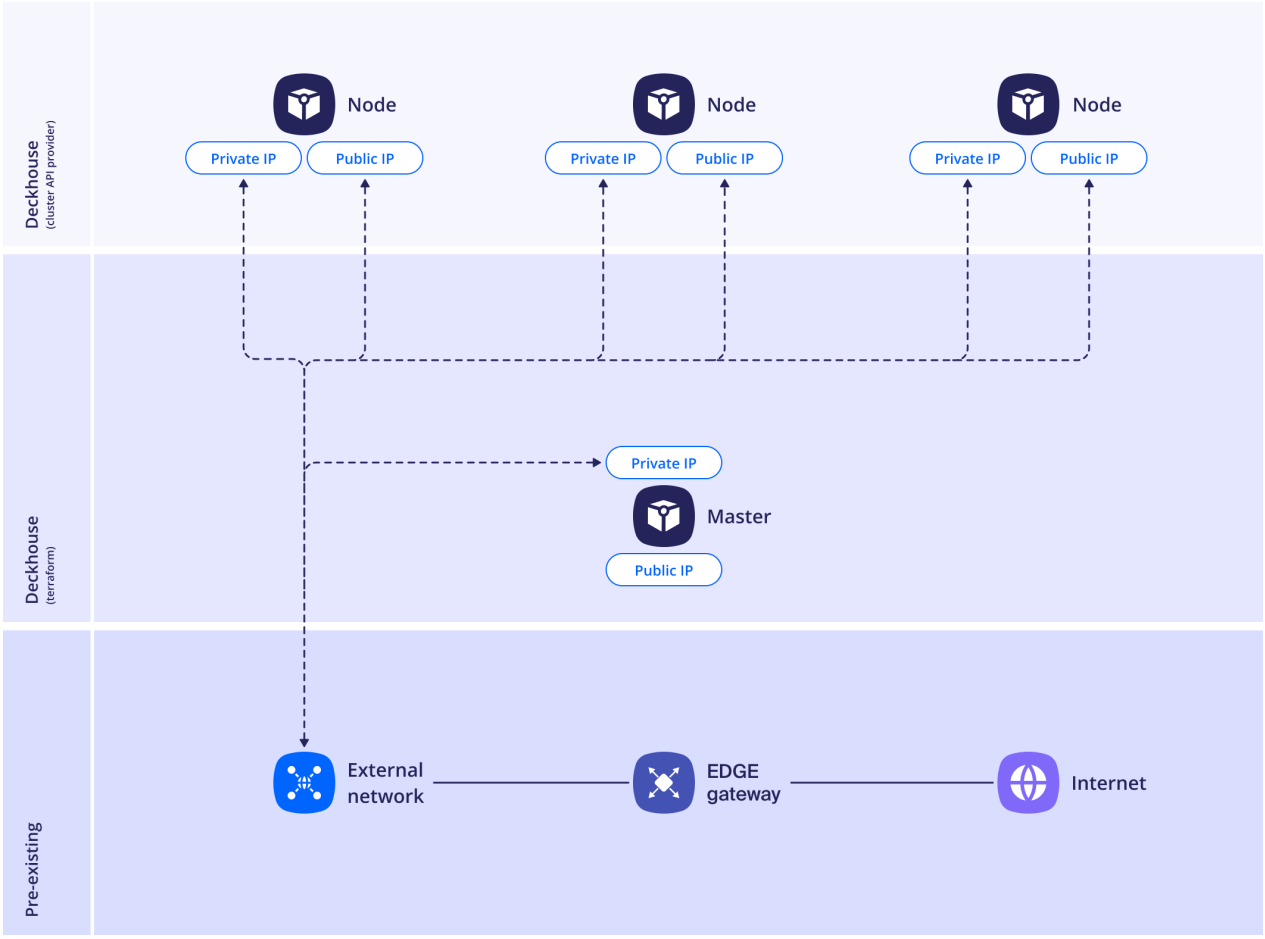
> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider - Basis Dynamix: Layouts

> ⚠ The module lifecycle stage: Preview

> ⚠ Changing `storagePolicy` recreates nodes of the `CloudEphemeral` type. The nodes that are recreated are listed in the module FAQ.

Standard



In this layout, the nodes are attached only to the external network. The nodes get DNS server addresses from the external network or from the `nameservers` parameter. If the external network doesn't provide DNS server addresses to the nodes, set them in `nameservers`. Otherwise, the nodes can't resolve names, and the installation stops at the preflight checks of the master node.

A new `nameservers` value applies only to the nodes created after the change.

Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
storagePolicy: storage_policy01
# Required if the external network doesn't provide DNS server addresses to the nodes.
nameservers: ["10.0.0.10"]
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

StandardWithInternalNetwork

In this layout, the nodes get DNS server addresses from the `nameservers` parameter through the ViNS.

Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
storagePolicy: storage_policy01
nodeNetworkCIDR: "10.241.32.0/24"
nameservers: ["10.0.0.10"]
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — GCP

The module lifecycle stage: General Availability

The `cloud-provider-gcp` module integrates Deckhouse Platform with Google Cloud Platform (GCP). It allows the `node-manager` module to use GCP resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-gcp` module:

- Managing GCP resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the GCP side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in GCP.
- Provisioning disks via the Persistent Disk CSI driver (`pd.csi.storage.gke.io`) and creating StorageClasses for GCP disk types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-google`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the GCPInstanceClass resource.
- Registering with `node-manager` so that GCPInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Third-party components

List of third-party software used in the `cloud-provider-gcp` module:

- Google Cloud Platform Cloud provider ccm/v33.4.0, ccm/v34.2.0, ccm/v35.0.5, ccm/v35.0.5, ccm/v35.0.5

  License: Apache License 2.0

  Cloud Controller Manager for Google Cloud Platform.

- Google Cloud Platform CSI driver v1.15.4, v1.15.4, v1.15.4, v1.15.4, v1.27.0

  License: Apache License 2.0

  CSI driver for Google Compute Engine persistent disks.

- Google Cloud Platform Terraform provider 4.85.0

  License: Mozilla Public License 2.0

  Terraform provider for Google Cloud Platform.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

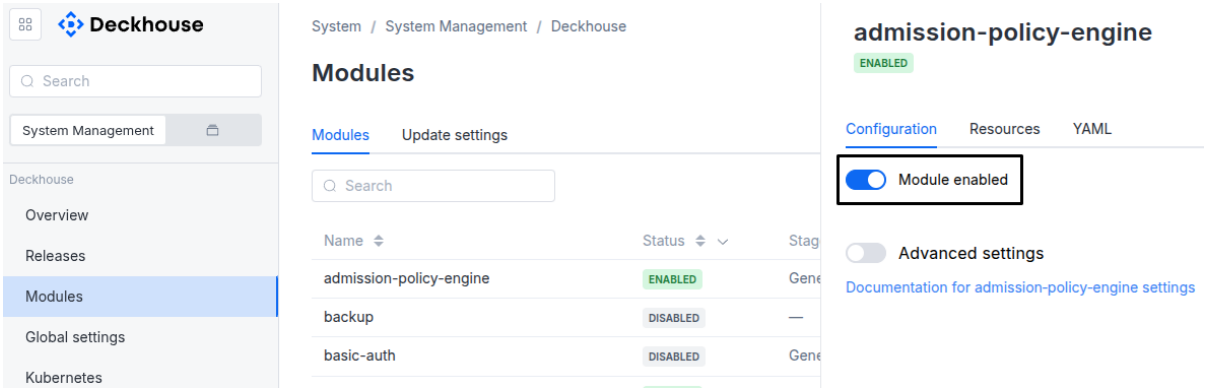
> The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy defined in the GCPClusterConfiguration struct. In most cases, you do not have to configure the module manually.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-gcp` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-gcp
```

- **Using ModuleConfig `cloud-provider-gcp`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-gcp` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-gcp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-gcp
spec:
  enabled: true
```

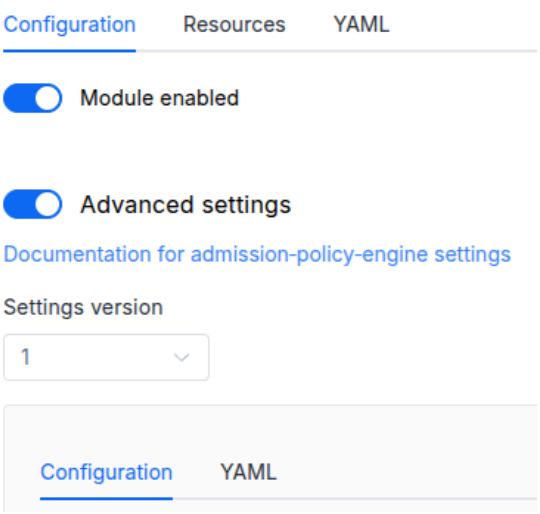
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-gcp` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `cloud-provider-gcp` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-gcp`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-gcp` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-gcp
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-gcp` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-gcp`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-gcp
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

You can configure the number and parameters of provisioning machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class’s name for the above group of nodes (the `cloudInstances.ClassReference` parameter of NodeGroup). In the case of the GCP cloud provider, the instance class is the `GCPInstanceClass` custom resource that stores specific parameters of the machines.

The module automatically creates StorageClasses that cover all the available disk types in GCP:

Type	Replication	StorageClass Name
standard	none	pd-standard-not-replicated
standard	regional	pd-standard-replicated
balanced	none	pd-balanced-not-replicated
balanced	regional	pd-balanced-replicated
ssd	none	pd-ssd-not-replicated
ssd	regional	pd-ssd-replicated

Also, it can filter out the unnecessary StorageClasses (you can do this via the `exclude` parameter).

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - existing StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module (in accordance with the order listed in the table above).

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: pd-ssd-not-replicated
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster;

      Example:

      ```
      exclude:
      - pd-standard.*
      - pd-ssd-replicated
      ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

GCPClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in GCP.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**
  string

  Required value

  Allowed values: `GCPClusterConfiguration`

- **labels**
  object

  A list of labels to attach to cluster resources.

  Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

  You can learn more about the labels in the official documentation.

  Format — `key: value`.

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  `Standard` - set Cloud NAT mode. More info...

  `WithoutNAT` - a dedicated VPC is created for the cluster. All cluster nodes have public IP addresses. More info...

  Allowed values: `Standard`, `WithoutNAT`

- **masterNodeGroup**
  object

  Required value

  Parameters of the master’s NodeGroup.

> **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

- **masterNodeGroup.instanceClass**
  object

  Required value

  Partial contents of the GCPInstanceClass fields.

  - **masterNodeGroup.instanceClass.additionalLabels**
    object

    Additional labels.

    For details on labels, refer to the Google Cloud documentation.

    Example:

    ```
    project: cms-production
    severity: critical
    ```

  - **masterNodeGroup.instanceClass.additionalNetworkTags**
    array of strings

    The list of additional network tags.

    You can use network tags to apply firewall rules to instances. The detailed description of network tags is available in the Google Cloud documentation.

  - **masterNodeGroup.instanceClass.disableExternalIP**
    boolean

    Defines whether to disable external IP for an instance or not.

    This parameter is only available for the `Standard` layout.

    True means that nodes do not have public addresses and connect to the Internet over `CloudNAT`;

    False means that static public addresses are created for nodes, they are also used for One-to-one NAT.;

    Default: `true`

    Allowed values: `true`, `false`

  - **masterNodeGroup.instanceClass.diskSizeGb**
    integer

    Instance disk size in gibibytes.

    Example:

    ```
    diskSizeGb: 50
    ```

  - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
    integer

    Etcd disk size in gibibytes.

    Default: `20`

    Example:

    ```
    etcdDiskSizeGb: 20
    ```

  - **masterNodeGroup.instanceClass.image**
    string

    Required value

    Image to use while provisioning GCP servers.

    You can find a list of available images in the provider documentation.

    The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

    Example:

```
image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

- **masterNodeGroup.instanceClass.machineType**
  string

  Required value

  Machine type of GCP instance.

  > **Caution!** Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the GCPInstanceClass fields.

    - **nodeGroups.instanceClass.additionalLabels**
      object

      Additional labels.

      For details on labels, refer to the Google Cloud documentation.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **nodeGroups.instanceClass.additionalNetworkTags**
      array of strings

      The list of additional network tags.

      You can use network tags to apply firewall rules to instances. The detailed description of network tags is available in the Google Cloud documentation.

    - **nodeGroups.instanceClass.disableExternalIP**
      boolean

      Defines whether to disable external IP for an instance or not.

      This parameter is only available for the `Standard` layout.

      True means that nodes do not have public addresses and connect to the Internet over

`CloudNAT`;

False means that static public addresses are created for nodes, they are also used for One-to-one NAT.;

Default: `true`

Allowed values: `true`, `false`

- **nodeGroups.instanceClass.diskSizeGb**
  integer

  Instance disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 50
  ```

- **nodeGroups.instanceClass.image**
  string

  Required value

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

  ```
  image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
  ```

- **nodeGroups.instanceClass.machineType**
  string

  Required value

  Machine type of GCP instance.

  > **Caution!** Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**
  object

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta).

  - **nodeGroups.nodeTemplate.labels**
    object

    The same as the `metadata.labels` standard (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta).

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object(https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#taint-v1-core).

> **i** Available fields: `effect`, `key`, and `values`.

        - **nodeGroups.nodeTemplate.taints.effect**
          string

          Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
        - **nodeGroups.nodeTemplate.taints.key**
          string
        - **nodeGroups.nodeTemplate.taints.value**
          string
    - **nodeGroups.replicas**
      integer

      Required value

      The number of nodes to create.
    - **nodeGroups.zones**
      array of strings

      A limited set of zones in which master nodes can be created.

      The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.
- **peeredVPCs**
  array of strings

  A list of GCP VPC networks to peer with the cluster network.

  The service account must have access to all the VPCs listed. You have to configure the peering connection manually if no access is available.
- **provider**
  object

  Required value

  Parameters for connecting to the GCP API.
    - **provider.region**
      string

      Required value

      The name of the region where instances will be provisioned.
    - **provider.serviceAccountJSON**
      string

      Required value

      A key to the Service Account with Project Admin privileges (`service account key`) in the JSON format.

      How to create it a `service account key`.
- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, from any.
- **sshKey**
  string

  Required value

  A public key to access nodes as `user`.
- **standard**
  object

  Settings for the `Standard` layout.
    - **standard.cloudNATAddresses**
      array of strings

      A list of public static IP addresses for `Cloud NAT`.

- **subnetworkCIDR**
  string

  A subnet to use for cluster nodes.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withoutNAT**
  object

  Settings for the `WithoutNAT` layout.

- **zones**
  array of strings

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> The module lifecycle stage: General Availability

> To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.

> The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:
>
> - `cloud-config.service`;
> - `cloud-final.service`;
> - `cloud-init.service`.

You need to create a service account so that Deckhouse Platform can manage resources in the Google Cloud. Below is a brief sequence of steps to create a service account. If you need detailed instructions, you can find them in the provider's documentation.

> The created `service account key` cannot be restored, you can only delete and create a new one.

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> Starting with DP version 1.77, GCP uses the `cilium` CNI by default for new clusters. Existing clusters keep the current CNI configuration.
>
> New clusters require Linux kernel version `5.8` or newer on all nodes. Make sure firewalls or security groups allow inter-node UDP traffic for Cilium VXLAN. For more information about Cilium requirements, limitations, and behavior, see the installation requirements, Network interaction of the platform components, and the `cni-cilium` module documentation.

Setup using Google Cloud Console

Follow this link, select your project and create a new service account or select an existing one.

The account must be assigned several necessary roles:

```
Compute Admin
Service Account User
Network Management Admin
```

You can add roles when creating a service account or edit them here.

To create a `service account key` in JSON format, click on three vertical dots in the Actions column and select `Manage keys`. Next, click on `Add key` -> `Create new key` -> `Key type` -> `JSON`.

Setup using gcloud CLI

To configure via the command line interface, follow these steps:

1. Export environment variables:

   ```
   export PROJECT_ID=sandbox
   export SERVICE_ACCOUNT_NAME=deckhouse
   ```

2. Select a project:

```
gcloud config set project $PROJECT_ID
```

3. Create a service account:

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

4. Connect roles to the service account:

```
for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
   --role=${role}; done
```

   List of roles required:

```
roles/compute.admin
roles/iam.serviceAccountUser
roles/networkmanagement.admin
```

5. Verify service account roles:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
     --filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Create a `service account key`:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
     ~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

GCPInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of GCP Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalDisks**
    array of objects

    A list of additional disks to attach to the instance.

    Useful for storage workloads such as LINSTOR, Ceph, NFS, and similar.

    Example:

```
additionalDisks:
- size: 100
  type: pd-ssd
```

- **spec.additionalDisks.autoDelete**
  boolean

  Automatically delete the disk when the instance is deleted.

  Default: `true`

- **spec.additionalDisks.provisionedIops**
  integer

  Provisioned IOPS for the disk.

  Required for `pd-extreme` and `hyperdisk-extreme` disk types.

  Allowed values: `1 <= X`

- **spec.additionalDisks.provisionedThroughput**
  integer

  Provisioned throughput in MiB/s for the disk.

  Applicable for `hyperdisk-balanced` and `hyperdisk-throughput` disk types.

  Allowed values: `1 <= X`

- **spec.additionalDisks.size**
  integer

  Required value

  Disk size in gigabytes (GB).

  Allowed values: `1 <= X`

  Example:

  ```
  size: 100
  ```

- **spec.additionalDisks.type**
  string

  Required value

  Disk type.

  - `pd-standard` — standard Persistent Disk (HDD).
  - `pd-balanced` — balanced Persistent Disk (SSD).
  - `pd-ssd` — SSD Persistent Disk.
  - `pd-extreme` — extreme Persistent Disk; requires `provisionedIops`.
  - `hyperdisk-balanced` — Hyperdisk with balanced performance.
  - `hyperdisk-throughput` — Hyperdisk optimized for throughput.
  - `hyperdisk-extreme` — Hyperdisk with maximum performance; requires `provisionedIops`.

  Allowed values: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

  Example:

  ```
  type: pd-ssd
  ```

- **spec.additionalLabels**
  object

  Additional labels.

  More info...

  Example:

  ```
  additionalLabels:
    project: cms-production
    severity: critical
  ```

- **spec.additionalNetworkTags**
  array of strings

  The list of additional tags.

  For example, you can use tags to apply firewall rules to instances. The detailed description of network tags is available in the official documentation.

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `machineType` parameter).

Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`a2-highgpu-8g`, `c2-standard-8`, etc.) is embedded in Deckhouse.

- **spec.capacity.cpu**
  integer or string

  Required value

  Node vCPU resources.

  Pattern: `^[0-9]+m?$`

  Example:

  ```
  cpu: 1000m
  ```

- **spec.capacity.memory**
  integer or string

  Required value

  Node memory resources.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  memory: 1500Mi
  ```

- **spec.disableExternalIP**
  boolean

  Defines whether to disable external ip for an instance or not. True means that an instance is created without external ip

- **spec.diskSizeGb**
  integer

  Instance’s root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 20
  ```

- **spec.diskType**
  string

  Instance’s disk type.

  Default: `pd-standard`

  Allowed values: `pd-standard`, `pd-ssd`

  Example:

  ```
  diskType: pd-ssd
  ```

- **spec.enableNestedVirtualization**
  boolean

  Enables nested virtualization on the instance.

  Required for running virtual machine workloads (e.g., KVM-based VMs) inside GCP instances.

  Warning. Only supported on specific machine types. See the GCP documentation for the list of compatible types.

  Default: `false`

- **spec.image**
  string

  Not required value.

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

```
image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

- **spec.machineType**
  string

  Required value

  Machine type of GCP instance.

  Caution! Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **spec.preemptible**
  boolean

  Order preemptible virtual machines (preemptible instances).

  Google Cloud can interrupt such a machine at any time.

  For more information about preemptible virtual machines, see the provider documentation.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of GCP Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalDisks**
    array of objects

    A list of additional disks to attach to the instance.

    Useful for storage workloads such as LINSTOR, Ceph, NFS, and similar.

    Example:

    ```
    additionalDisks:
    - size: 100
      type: pd-ssd
    ```

    - **spec.additionalDisks.autoDelete**
      boolean

      Automatically delete the disk when the instance is deleted.

      Default: `true`

    - **spec.additionalDisks.provisionedIops**
      integer

      Provisioned IOPS for the disk.

      Required for `pd-extreme` and `hyperdisk-extreme` disk types.

      Allowed values: `1 <= X`

    - **spec.additionalDisks.provisionedThroughput**
      integer

      Provisioned throughput in MiB/s for the disk.

      Applicable for `hyperdisk-balanced` and `hyperdisk-throughput` disk types.

      Allowed values: `1 <= X`

    - **spec.additionalDisks.size**
      integer

      Required value

      Disk size in gigabytes (GB).

      Allowed values: `1 <= X`

Example:

```
size: 100
```

- **spec.additionalDisks.type**
  string

  Required value

  Disk type.

  - `pd-standard` — standard Persistent Disk (HDD).
  - `pd-balanced` — balanced Persistent Disk (SSD).
  - `pd-ssd` — SSD Persistent Disk.
  - `pd-extreme` — extreme Persistent Disk; requires `provisionedIops`.
  - `hyperdisk-balanced` — Hyperdisk with balanced performance.
  - `hyperdisk-throughput` — Hyperdisk optimized for throughput.
  - `hyperdisk-extreme` — Hyperdisk with maximum performance; requires `provisionedIops`.

  Allowed values: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

  Example:

```
type: pd-ssd
```

- **spec.additionalLabels**
  object

  Additional labels.

  More info...

  Example:

```
additionalLabels:
  project: cms-production
  severity: critical
```

- **spec.additionalNetworkTags**
  array of strings

  The list of additional tags.

  For example, you can use tags to apply firewall rules to instances. The detailed description of network tags is available in the official documentation.

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `machineType` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`a2-highgpu-8g`, `c2-standard-8`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

```
cpu: 1000m
```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

```
memory: 1500Mi
```

- **spec.disableExternalIP**
  boolean

  Defines whether to disable external ip for an instance or not. True means that an instance is created without external ip

- **spec.diskSizeGb**
  integer

  Instance’s root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 20
  ```

- **spec.diskType**
  string

  Instance’s disk type.

  Default: `pd-standard`

  Allowed values: `pd-standard`, `pd-ssd`

  Example:

  ```
  diskType: pd-ssd
  ```

- **spec.enableNestedVirtualization**
  boolean

  Enables nested virtualization on the instance.

  Required for running virtual machine workloads (e.g., KVM-based VMs) inside GCP instances.

  Warning. Only supported on specific machine types. See the GCP documentation for the list of compatible types.

  Default: `false`

- **spec.image**
  string

  Not required value.

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

  ```
  image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
  ```

- **spec.machineType**
  string

  Required value

  Machine type of GCP instance.

  Caution! Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **spec.preemptible**
  boolean

  Order preemptible virtual machines (preemptible instances).

  Google Cloud can interrupt such a machine at any time.

  For more information about preemptible virtual machines, see the provider documentation.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How do I create a cluster?

1. Set up a cloud environment.
2. Enable the module or pass the `--extra-config-map-data base64_encoding_of_custom_config` flag with the module parameters to the `install.sh` script.
3. Create one or more GCPInstanceClass custom resources.
4. Create one or more NodeGroup custom resources for managing the number and the process of provisioning machines in the cloud.

Adding CloudStatic nodes to a cluster

For the VMs, you want to add to a cluster as nodes, add a `Network Tag` similar to the cluster prefix.

You can find out `prefix` using the command:

```
d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
```

If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
  | base64 -d | grep prefix
```

How to create preemptible nodes in GCP?

Preemptible virtual machines in Google Cloud are called preemptible instances. To make a node group order them, set the `preemptible` parameter of the GCPInstanceClass resource to `true`.

Example:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: worker-preempt
spec:
  machineType: n1-standard-4
  diskSizeGb: 30
  preemptible: true
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-preempt
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: GCPInstanceClass
      name: worker-preempt
    minPerZone: 2
    maxPerZone: 4
    zones:
      - europe-west4-a
```

Along with this parameter, Deckhouse Platform (DP) disables the automatic restart of the virtual machine and makes it stop during host maintenance.

Limitations of preemptible nodes in GCP

Consider the following limitations:

- The `preemptible` parameter applies only to CloudEphemeral node groups, because GCPInstanceClass applies only to them.
- DP does not track Google Cloud notices about an upcoming preemption and does not drain the node in advance. When Google Cloud stops the instance, the node switches to the `NotReady` state, and the group orders a new node in place of the lost one. Until then, the pods of the stopped node may appear in the Kubernetes API with the `Running` status even though they are no longer available.
- Place only the workload that tolerates a sudden node loss on preemptible nodes.

⚠ Changing the `preemptible` parameter in an existing GCPInstanceClass causes all nodes of the corresponding group to be recreated.

How do I replace the OS image on the nodes?

The OS image is set by the `image` parameter of the GCPInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the GCPClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?”.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

The module lifecycle stage: General Availability

An example of the `GCPInstanceClass` custom resource

Below is a simple example of custom resource `GCPInstanceClass` configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: test
spec:
  machineType: n1-standard-1
```

Enabling nested virtualization

To run virtual machine workloads (e.g., KVM-based VMs) inside GCP instances, enable nested virtualization.

Only specific machine types support nested virtualization. See the GCP documentation for the list of compatible types.

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  enableNestedVirtualization: true
```

Adding additional disks

To attach additional disks to instances (for example, for LINSTOR, Ceph, NFS storage nodes, and similar solutions), specify them in the `additionalDisks` parameter:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
  - size: 200
    type: pd-ssd
  - size: 500
    type: pd-standard
    autoDelete: true
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

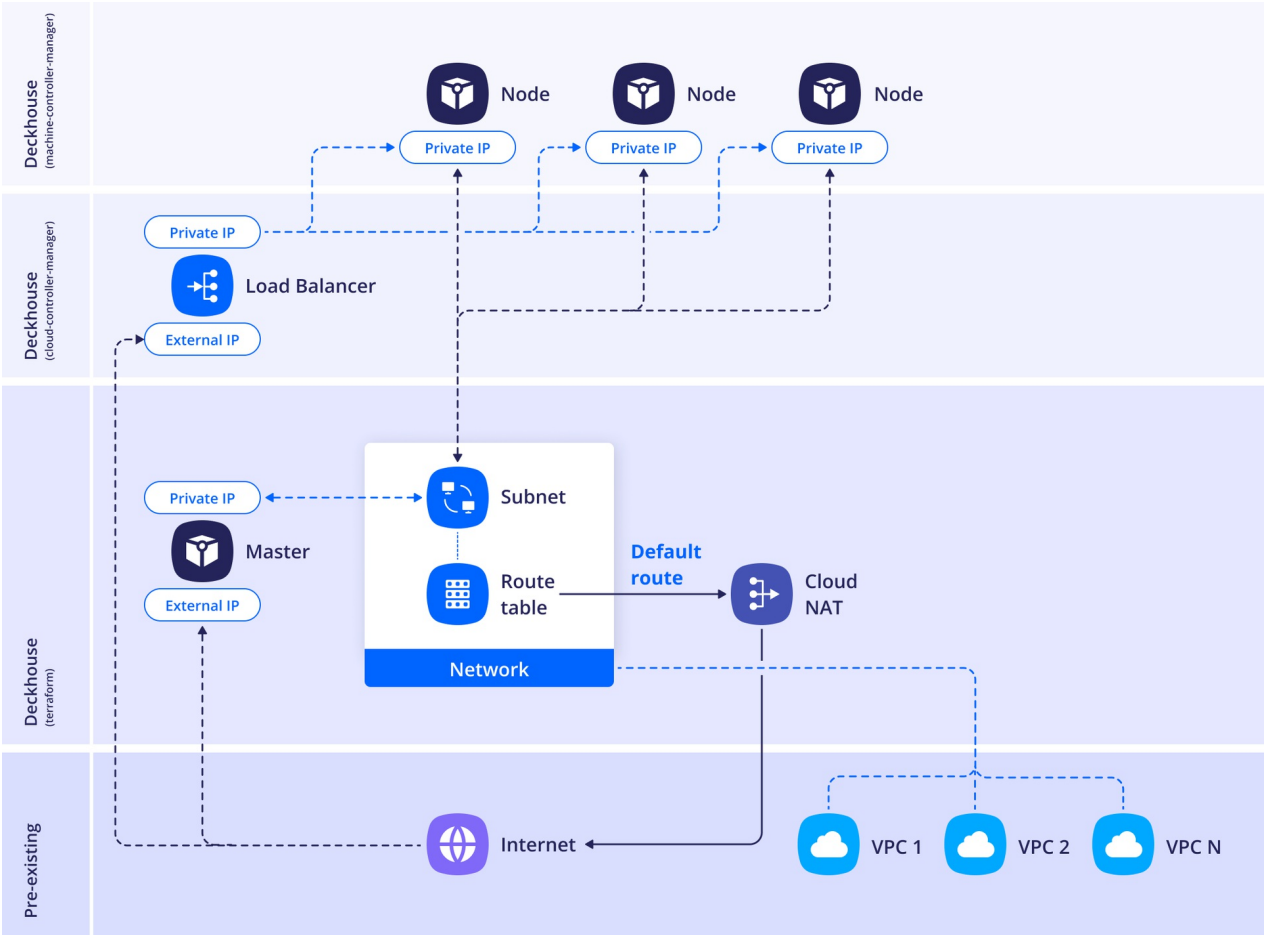
Layouts

> The module lifecycle stage: General Availability

Two layouts are supported. Below is more information about each of them.

Standard

- A separate VPC with Cloud NAT is created for the cluster.
- Nodes in the cluster do not have public IP addresses.
- Public IP addresses can be allocated to master and static nodes:
  - In this case, one-to-one NAT is used to translate the public IP address to the node's IP address (note that CloudNAT is not used in such a case).
- If the master does not have a public IP, then an additional instance with a public IP (aka bastion host) is required for installation tasks and accessing the cluster.
- Peering can also be configured between the cluster VPC and other VPCs.



When a cluster is created, the module creates the following firewall rules in the cluster VPC:

- `<CLUSTER_PREFIX>-ssh-and-ping` — allow incoming traffic over the ICMP and TCP (port `22`) protocols to nodes with the `<CLUSTER_PREFIX>` network tag from CIDRs listed in `sshAllowList` (default `0.0.0.0/0`)

- `<CLUSTER_PREFIX>-intercommunication` — allow any traffic between nodes with the `<CLUSTER_PREFIX>` network tag, and from the pod subnet (`podSubnetCIDR`)

Apply custom firewall rules to nodes via additional network tags (`additionalNetworkTags`):

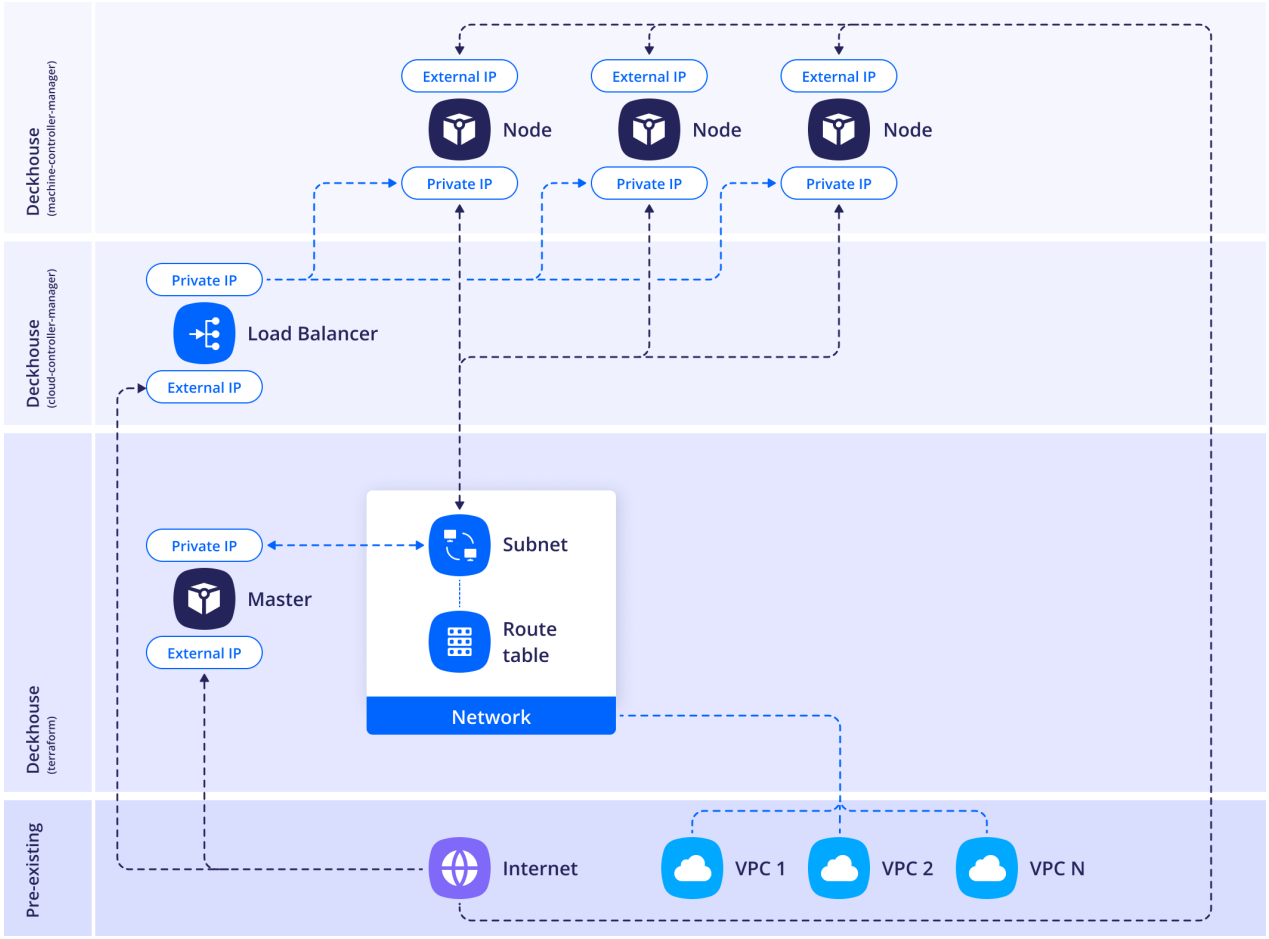
- for master nodes — in the `masterNodeGroup.instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalNetworkTags` parameter of the GCPInstanceClass resource

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Optional, compute address names from this list are used as addresses for Cloud NAT.
  cloudNATAddresses:
  - example-address-1
  - example-address-2
subnetworkCIDR: 10.0.0.0/24       # Required.
peeredVPCs:
# Optional, list of GCP VPC Networks with which Kubernetes VPC Network will be peered.
- default
sshKey: "<SSH_PUBLIC_KEY>"  # Required.
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                       # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    disableExternalIP: false       # Optional, by default master has externalIP.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                       # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    disableExternalIP: true        # Optional, by default nodes do not have externalIP.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: static
provider:
  region: europe-west4           # Required.
  serviceAccountJSON: |          # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

WithoutNAT

- A dedicated VPC is created for the cluster; all cluster nodes have public IP addresses.
- Peering can be configured between the cluster VPC and other VPCs.



When a cluster is created, the module creates the following firewall rules in the cluster VPC:

- `<CLUSTER_PREFIX>-ssh-and-ping` — allow incoming traffic over the ICMP and TCP (port `22`) protocols to nodes with the `<CLUSTER_PREFIX>` network tag from CIDRs listed in `sshAllowList` (default `0.0.0.0/0`)
- `<CLUSTER_PREFIX>-intercommunication` — allow any traffic between nodes with the `<CLUSTER_PREFIX>` network tag, and from the pod subnet (`podSubnetCIDR`)

Apply custom firewall rules to nodes via additional network tags (`additionalNetworkTags`):

- for master nodes — in the `masterNodeGroup.instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalNetworkTags` parameter of the GCPClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalNetworkTags` parameter of the GCPInstanceClass resource

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional, list of GCP VPC Networks with which Kubernetes VPC Network will be peered.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: static
provider:
  region: europe-west4            # Required.
  serviceAccountJSON: |           # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — Huawei Cloud

Included in extensions: **On-prem IaaS**

The module lifecycle stage: General Availability

The `cloud-provider-huaweicloud` module integrates Deckhouse Platform with Huawei Cloud. It allows the `node-manager` module to use Huawei Cloud resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-huaweicloud` module:

- Managing Huawei Cloud resources via `cloud-controller-manager`:
  - updates instance and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in the cloud;
  - creates load balancers (ELB) for Services of the LoadBalancer type.

- Provisioning disks via the EVS CSI driver (`evs.csi.huaweicloud.com`) so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-huaweicloud`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the HuaweiCloudInstanceClass resource.
- Registering with `node-manager` so that HuaweiCloudInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Third-party components

List of third-party software used in the `cloud-provider-huaweicloud` module:

- Huawei Cloud provider v0.26.9

  License: Apache License 2.0

  Cloud Controller Manager for Huawei Cloud.

- Huawei CSI driver evs-0.1.11

  License: Apache License 2.0

  CSI driver for Huawei Cloud EVS and SFS.

- Huawei Terraform provider 1.69.0

  License: Mozilla Public License 2.0

  Terraform provider for Huawei Cloud.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Included in extensions: **On-prem IaaS**

> The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in Huawei Cloud.

List of required Huawei Cloud services

A list of services required for Deckhouse Platform to work in Huawei Cloud:

Service	API version
Identity	v3
Compute	v2.1
Network (VPC)	v1/v2
Network (OpenStack Neutron API)	v2.0
Block Storage	v3
Load Balancing (ELB) *	v3

* Required if load balancers need to be provisioned in the cluster.

The module has 1 alert.

The module does not have any settings.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

HuaweiCloudClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Huawei Cloud.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

Example:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
standard:
  externalNetworkName: external-network
provider:
  domainName: "<DOMAIN_NAME>"
  region: eu-3
  accessKey: "<USERNAME>"
  secretKey: "<PASSWORD>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    flavorName: m1.large
    rootDiskSize: 50
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **kind**
  string

  Required value

  Allowed values: `HuaweiCloudClusterConfiguration`

- **layout**
  string

  Required value

  The layout name.

  For details about possible provider layouts, refer to Layouts.

- **masterNodeGroup**

  Required value

  The definition of a NodeGroup for master nodes.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**

object

Required value

Partial contents of the fields of the HuaweiCloudInstanceClass.

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in gigabytes.

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.flavorName**
  string

  Required value

  The flavor of Huawei Cloud servers.

- **masterNodeGroup.instanceClass.imageName**
  string

  Required value

  The image to use while provisioning Huawei Cloud servers.

  For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

- **masterNodeGroup.instanceClass.rootDiskSize**
  integer

  The size of a root disk in gigabytes.

  This parameter also affects the type of a root disk.

  Default: `50`

  Example:

  ```
  rootDiskSize: 50
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.serverGroup**
  object

  The ServerGroup object groups instances together. The instances in the group are placed on the same hypervisor (affinity) or different hypervisors (anti-affinity). This allows you to increase the fault tolerance of the cluster.

  - **masterNodeGroup.serverGroup.policy**
    string

    Required value

    The policy that determines how instances are distributed among hypervisors.

    - `AntiAffinity`: Instances are placed on different hypervisors. This policy is managed by Deckhouse.

    Allowed values: `AntiAffinity`

- **masterNodeGroup.volumeTypeMap**
  object

  Required value

  A dictionary of disk types for storing etcd data and Kubernetes configuration files.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <DISK TYPE>` (see the example).

  If the `rootDiskSize` parameter is specified, the same disk type will be used for the VM's boot drive.

  We recommend using the fastest disks provided by the provider in all cases.

  If the value specified in `replicas` exceeds the number of elements in the dictionary, the master nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if `replicas: 5`, then master-0, master-2, master-4 will have the `fast-eu-1a` disk type, while master-1, master-3 will have the `fast-eu-1b` disk type.

Example:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields.

    - **nodeGroups.instanceClass.additionalNetworks**
      array of strings

      Not required value.

      List of network IDs (subnets) for secondary NICs.

    - **nodeGroups.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in gigabytes.

      Default: `10`

      Example:

      ```
      etcdDiskSizeGb: 10
      ```

    - **nodeGroups.instanceClass.flavorName**
      string

      Required value

      The flavor of Huawei Cloud servers.

    - **nodeGroups.instanceClass.imageName**
      string

      Required value

      The image to use while provisioning Huawei Cloud servers.

      For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

    - **nodeGroups.instanceClass.mainNetwork**
      string

      Not required value.

      Primary network ID for the VM’s main NIC. **By default**, the value from ‘HuaweiCloudDiscoverData’ is used.

    - **nodeGroups.instanceClass.rootDiskSize**
      integer

      The size of a root disk in gigabytes.

      This parameter also affects the type of a root disk.

      Default: `50`

      Example:

      ```
      rootDiskSize: 50
      ```

  - **nodeGroups.name**
    string

    Required value

    The name of the NodeGroup to use for generating node names.

  - **nodeGroups.nodeTemplate**

    Parameters of Node objects in Kubernetes to add after registering the node.

    - **nodeGroups.nodeTemplate.annotations**
      object

      The same as the `metadata.annotations` standard field.

      Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**
  object

  A list of labels to attach to all cluster resources, if they support it.

  The same as the `metadata.labels` standard field.

  If you change a label on a running cluster, recreate all virtual machines after the changes come into effect.

  Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeGroups.volumeTypeMap**
  object

  A dictionary mapping availability zones to disk types for the group's worker nodes.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <DISK TYPE>` (see the example).

  If the `rootDiskSize` parameter is specified, the same disk type is used for the VM's boot disk.

  We recommend using the fastest disk type offered by the provider whenever possible.

  If `volumeTypeMap` is not specified for a node group, the `masterNodeGroup`'s `volumeTypeMap` is used as a fallback for that group.

  If the replicas value exceeds the number of entries in the dictionary, assignment wraps around from the beginning. For example, if `replicas: 5` for the `front` group and the dictionary lists `fast-eu-1a` first and `fast-eu-1b` second, then front-0, front-2, front-4 will use `fast-eu-1a`, while front-1, front-3 will use `fast-eu-1b`.

  Example:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **provider**
  object

  Required value

  Huawei Cloud API connection settings.

  These settings match the ones in the `connection` field of the cloud-provider-huaweicloud module.

  - **provider.accessKey**
    string

Required value

The Huawei Cloud access key.

- **provider.authURL**
  string

  The identity authentication URL.

- **provider.cloud**
  string

  Required value

  The endpoint of the cloud provider.

- **provider.domainName**
  string

  The account name of IAM to scope to.

- **provider.enterpriseProjectID**
  string

  The enterprise project ID.

- **provider.insecure**
  boolean

  Trust self-signed SSL certificates.

- **provider.projectID**
  string

  The project ID.

- **provider.region**
  string

  Required value

  The Huawei Cloud region where the cluster will be deployed.

- **provider.secretKey**
  string

  Required value

  The Huawei Cloud secret key.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.enableEIP**
    boolean

    Enable Elastic IP for the master nodes.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **standard.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster node network.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **standard.internalNetworkDNSServers**
    array of strings

    A list of addresses of the recursive DNSs for the internal cluster node network.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **standard.internalNetworkSecurity**
    boolean

Defines whether SecurityGroups must be configured for the internal network ports.

When enabled, DP creates a security group named after the cluster `prefix` with the following default inbound rules:

- TCP/22 (SSH) from `0.0.0.0/0` by default
- ICMP from `0.0.0.0/0`
- TCP NodePorts `30000-32767` from `0.0.0.0/0`

For details, see Huawei Cloud: Layouts.

Default: `true`

- **vpcPeering**
  object

  Settings for the `VpcPeering` layout.

  - **vpcPeering.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster node network.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **vpcPeering.internalNetworkDNSServers**
    array of strings

    A list of addresses of the recursive DNSs for the internal cluster node network.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **vpcPeering.internalNetworkSecurity**

    Defines whether SecurityGroups must be configured for the internal network ports.

  - **vpcPeering.subnet**
    string

    The name of the subnet in the VPC.

- **zones**
  array of strings

  Not required value.

  The globally restricted set of zones that this cloud provider works with.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> **i** The module lifecycle stage: General Availability

> ⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.

> ⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:
>
> - `cloud-config.service`;
> - `cloud-final.service`;
> - `cloud-init.service`.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Deckhouse Platform components interact with Huawei Cloud resources through the Huawei Cloud API. To configure this connection, you need to create a user in the Huawei Cloud IAM service and provide it with the necessary permissions.

Configuring IAM via the web interface

To configure IAM via the web interface, first create a new user group and assign the necessary permissions. Follow these steps:

1. Go to the “Identity and Access Management (IAM)” section.
2. Open the “User Groups” page and click “Create User Group”.
3. In the “Name” field, enter the group name (e.g., `deckhouse`).
4. Click “OK” to create the group.
5. Select the newly created group from the list.
6. On the “Permissions” tab, click “Authorize”.
7. Assign the following policies: “ECS Admin”, “VPC Administrator”, “NAT Admin”, “ELB FullAccess”, “DEW KeypairFullAccess”.
8. Click “Next”, then “OK”, and complete the setup by clicking “Finish”.

Then add a new user. Follow these steps:

1. Go to the “Users” page in the IAM section and click “Create User”.
2. In the “Username” field, enter the username (e.g., `deckhouse`).
3. Set “Access type” to “Programmatic access” and make sure “Management console access” is disabled.
4. Select “Access key” as the “Credential Type”.
5. Click “Next”.
6. Select the previously created user group.
7. Click “Create” to complete the user creation process.
8. Click “OK” to download the `Access Key ID` and `Secret Access Key`. Make sure to save these credentials in a secure location, as they will be needed to access the API.

JSON policies

Below are the contents of the policies in JSON format:

ECS Admin policy

```
{
"Version": "1.1",
"Statement": [
{
    "Action": [
    "ecs:*:*",
    "evs:*:get",
    "evs:*:list",
    "evs:volumes:create",
    "evs:volumes:delete",
    "evs:volumes:attach",
    "evs:volumes:detach",
    "evs:volumes:manage",
    "evs:volumes:update",
    "evs:volumes:use",
    "evs:volumes:uploadImage",
    "evs:snapshots:create",
    "vpc:*:get",
    "vpc:*:list",
    "vpc:networks:create",
    "vpc:networks:update",
    "vpc:subnets:update",
    "vpc:subnets:create",
    "vpc:ports:*",
    "vpc:routers:get",
    "vpc:routers:update",
    "vpc:securityGroups:*",
    "vpc:securityGroupRules:*",
    "vpc:floatingIps:*",
    "vpc:publicIps:*",
    "ims:images:create",
    "ims:images:delete",
    "ims:images:get",
    "ims:images:list",
    "ims:images:update",
    "ims:images:upload"
    ],
    "Effect": "Allow"
}
]
}
```

VPC Administrator policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "vpc:vpcs:*",
                "vpc:routers:*",
                "vpc:networks:*",
                "vpc:subnets:*",
                "vpc:ports:*",
                "vpc:privateIps:*",
                "vpc:peerings:*",
                "vpc:routes:*",
                "vpc:lbaas:*",
                "vpc:vpns:*",
                "ecs:*:get",
                "ecs:*:list",
                "elb:*:get",
                "elb:*:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

NAT Admin policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "nat:*:*",
                "vpc:*:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

DEW KeypairFullAccess policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kps:domainKeypairs:*",
                "ecs:serverKeypairs:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ELB FullAccess policy

```
{
  "Version": "1.1",
  "Statement": [
      {
          "Action": [
              "elb:*:*",
              "vpc:*:get*",
              "vpc:*:list*",
              "ecs:*:get*",
              "ecs:*:list*"
          ],
          "Effect": "Allow"
      }
  ]
}
```

Replacing the OS image

The OS image is set by the `imageName` parameter of the HuaweiCloudInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the HuaweiCloudClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> **i** The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

HuaweiCloudInstanceClass

Scope: Cluster

Version: v1

Huawei Cloud server group parameters used by `machine-controller-manager` (the node-manager module).

The CloudInstanceClass resource of the `node-manager` module refers to this resource.

Example:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Not required value.

    List of networks IDs (subnets) for secondary NICs.

  - **spec.flavorName**
    string

    Required value

    Flavor of Huawei Cloud servers.

    To get a list of all available flavors, run the following command: `cloud ECS ListFlavors`.

    For all non-master nodes, it’s recommended that you use a flavor that includes a local disk. If the cloud provider supports local disks, they’re usually faster and cheaper. The disadvantage of using such a flavor is the inability to migrate nodes between hypervisors.

    Example:

    ```
    flavorName: m1.medium
    ```

  - **spec.imageName**
    string

    Not required value.

    Image to use while provisioning HuaweiCloud servers.

    Use this command to get a list of available images: `cloud IMS ListImages`.

    For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

    By default, either the `HuaweiCloudDiscoveryData` value or `instances.imageName` is used.

    Example:

    ```
    imageName: ubuntu-18-04-cloud-amd64
    ```

  - **spec.mainNetwork**
    string

    Not required value.

    ID of the primary network for the VM’s main NIC.

    By default, the value from ‘HuaweiCloudDiscoverData’ is used.

  - **spec.rootDiskSize**
    integer

    Not required value.

    The size of a root disk in gigabytes.

    This parameter also affects the type of a root disk.

    Example:

    ```
    rootDiskSize: 20
    ```

- **spec.rootDiskType**
  string

  Not required value.

  The type of a root disk.

  The parameter is used to specify the type of a volume that will be used for the root disk.

  By default, the value from `HuaweiCloudDiscoveryData` is used.

  Example:

  ```
  rootDiskType: GPSSD
  ```

- **spec.securityGroups**
  array of strings

  Not required value.

  Security groups that will be applied to VM’s network ports in addition to security groups set in a cloud provider configuration.

  These security groups allow you to set firewall rules for provisioned instances.

  The `SecurityGroups` group may not be supported by the cloud provider.

  Example:

  ```
  securityGroups:
  - security-group-1
  - security-group-2
  ```

- **spec.serverGroupID**
  string

  Not required value.

  The server group to assign the machine to.

  Example:

  ```
  serverGroupID: server-group-1
  ```

- **spec.subnets**
  Deprecated
  array of strings

  Not required value.

  > **i** Use the additionalNetworks field instead.

  Paths to networks that the secondary VirtualMachines NICs will connect to.

  Example:

  ```
  subnets:
  - BGP-network-VLAN-3894
  - External-VLAN-3699
  ```

- **spec.vipAddress**
  string

  Not required value.

  If filled, enables Virtual IP for all nodes in this instance class.

  `Auto` or IPv4 address are expected.

  Pattern: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^Auto$`

  Example:

  ```
  vipAddress: Auto
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

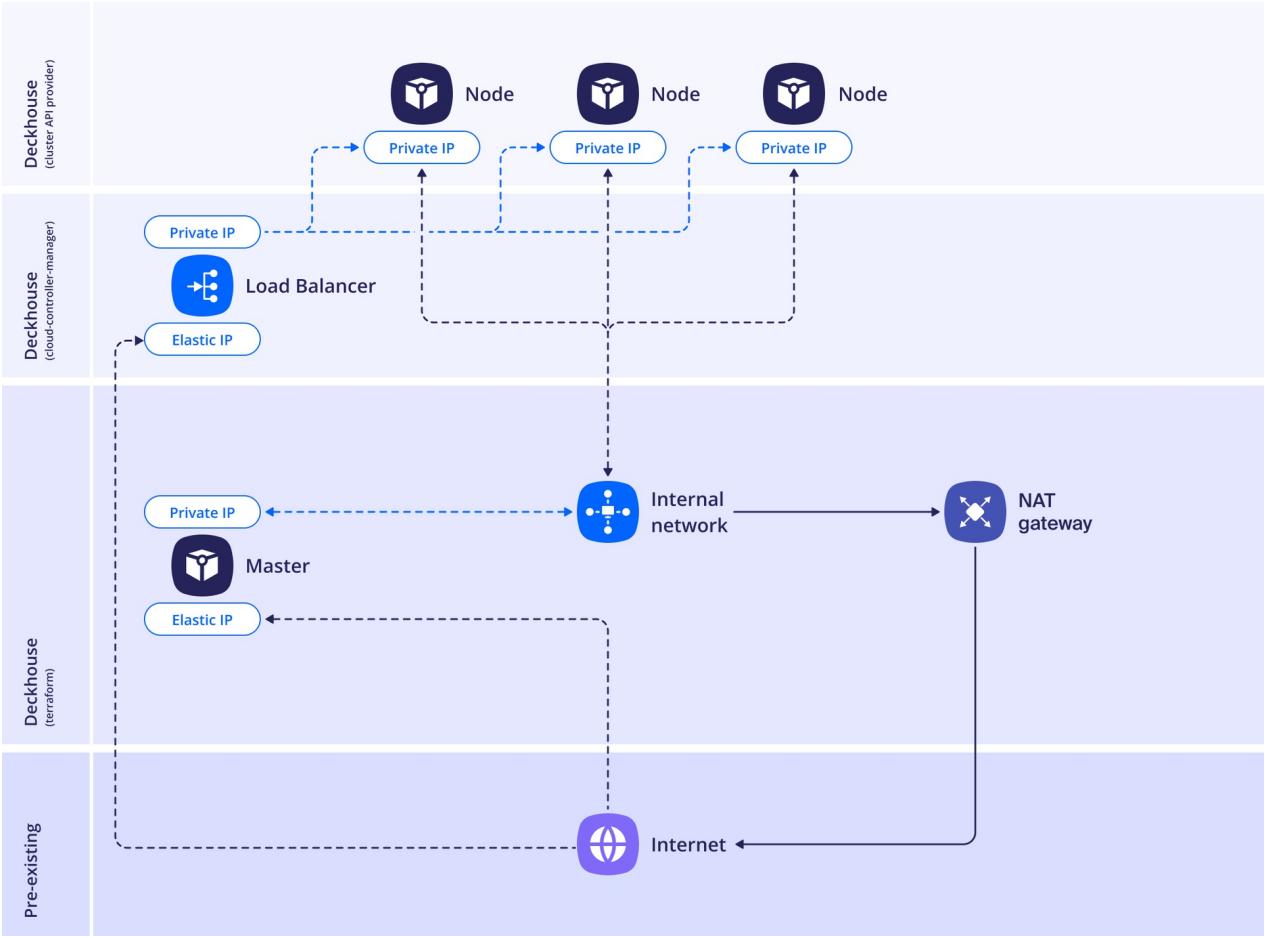
> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider - HuaweiCloud: Layouts

> The module lifecycle stage: General Availability

Standard

- An internal cluster network is created with a gateway to the public network.
- The elastic IP can be assigned to the master node.
- Nodes managed by the Cluster API do not have public IP addresses.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. The module creates the group named after the cluster prefix and assigns it to the nodes.

The following inbound rules will be created:

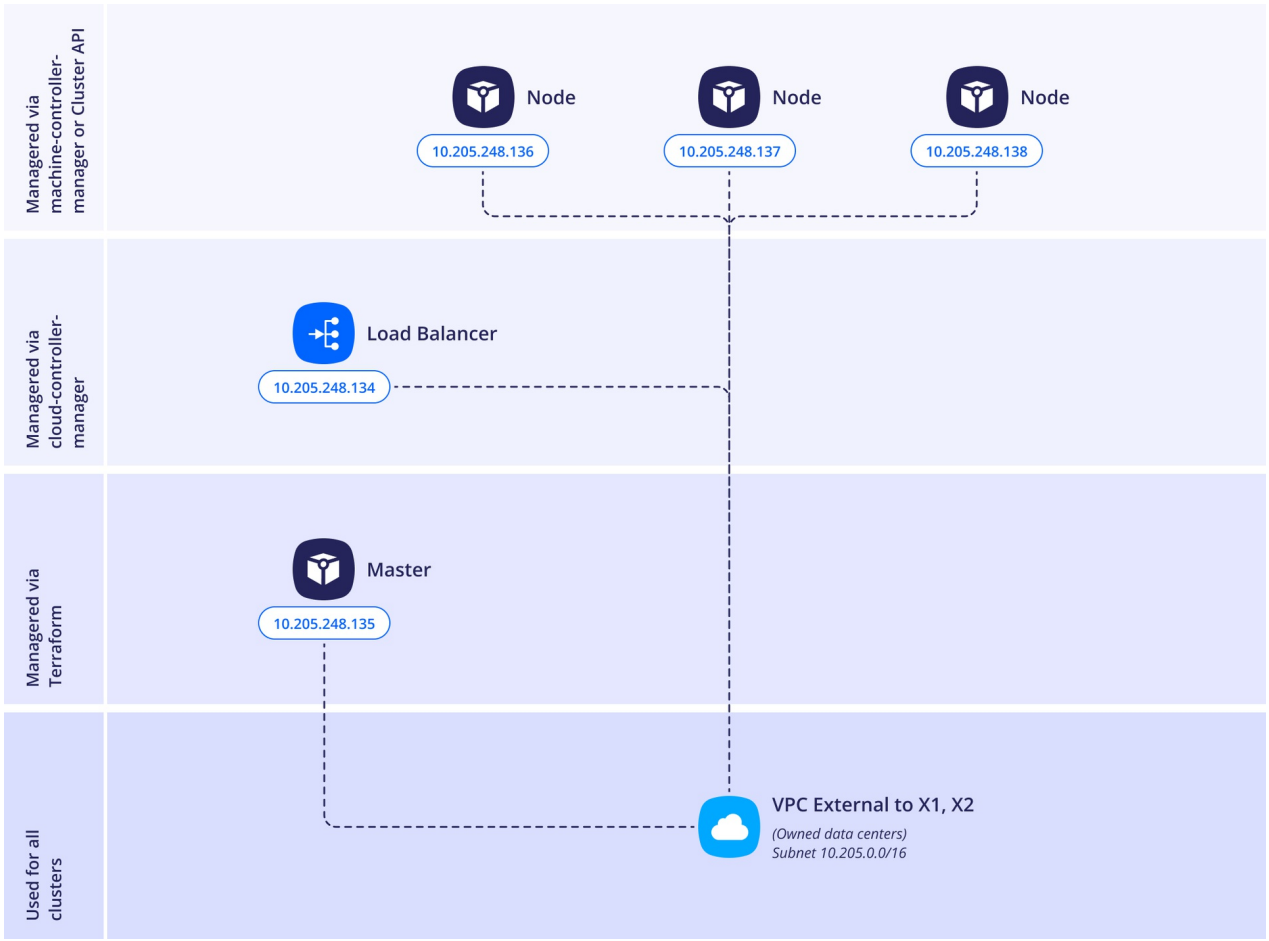
- allow incoming traffic over the TCP protocol on port `22` from `0.0.0.0/0`
- allow incoming traffic over the ICMP protocol from `0.0.0.0/0`
- allow incoming traffic over the TCP protocol on ports `30000` – `32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default

Attach custom security groups for CloudEphemeral nodes in the HuaweiCloudInstanceClass resource via `spec.securityGroups`. They are applied together with the group created by the module.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. The module creates the group named after the cluster prefix and assigns it to the nodes.

The following inbound rules will be created:

- allow incoming traffic over the TCP protocol on port `22` from `0.0.0.0/0`
- allow incoming traffic over the ICMP protocol from `0.0.0.0/0`
- allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default

Attach custom security groups for CloudEphemeral nodes in the HuaweiCloudInstanceClass resource via `spec.securityGroups`. They are applied together with the group created by the module.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — OpenStack

Included in extensions: **On-prem IaaS**

The module lifecycle stage: General Availability

The `cloud-provider-openstack` module integrates Deckhouse Platform with OpenStack-based clouds. It allows the `node-manager` module to use OpenStack resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-openstack` module:

- Managing OpenStack resources via `cloud-controller-manager`:
  - updates OpenStack server and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in OpenStack;
  - creates load balancers (Octavia) for Services of the LoadBalancer type.
- Provisioning block disks via the Cinder CSI driver (`cinder.csi.openstack.org`). Manila (filesystem) is not supported. The Cinder CSI driver supports re-authentication in OpenStack with service catalog refresh, which improves resilience for long-running pods with volumes.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-openstack`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM) or Cluster API (CAPI). Virtual machine parameters are set in the OpenStackInstanceClass resource.
- Registering with `node-manager` so that OpenStackInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used. The network mode depends on the `podNetworkMode` parameter.

⚠ The module is migrating CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI). Existing NodeGroups continue to use MCM, while new ones are created with CAPI by default. For migrating existing groups, see How to migrate node groups to Cluster API (CAPI).

Third-party components

List of third-party software used in the `cloud-provider-openstack` module:

- OpenStack Cloud provider v1.33.1, v1.34.1, v1.35.0, v1.36.0, v1.36.0

  License: Apache License 2.0

  Cloud Controller Manager for OpenStack.

- OpenStack CSI driver v1.33.1, v1.34.1, v1.35.0, v1.36.0, v1.36.0

  License: Apache License 2.0

  CSI driver for OpenStack Cinder.

- OpenStack Terraform provider 1.54.1

  License: Mozilla Public License 2.0

  Terraform provider for OpenStack.

- OpenStack Cluster API provider v0.14.8

  License: Apache License 2.0

  Infrastructure provider for Kubernetes Cluster API in OpenStack.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Included in extensions: **On-prem IaaS**

> The module lifecycle stage: General Availability

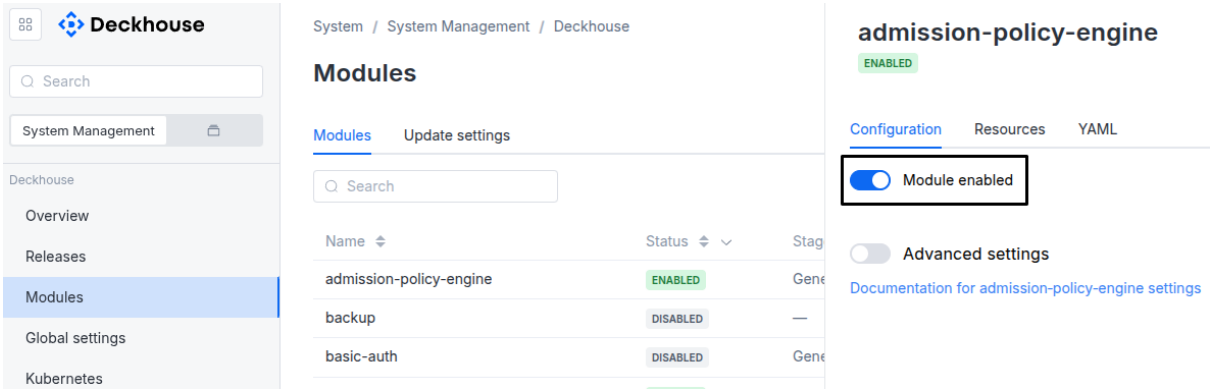
The module is automatically enabled for all cloud clusters deployed in OpenStack.

The module has 1 alert.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-openstack` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-openstack
```

- **Using ModuleConfig `cloud-provider-openstack`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-openstack` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-openstack`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  enabled: true
```

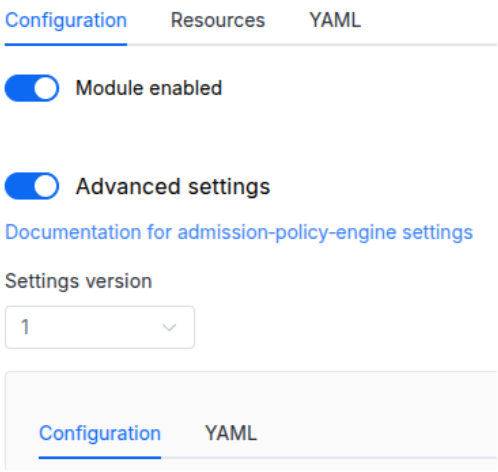
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-openstack` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cloud-provider-openstack` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-openstack`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-openstack` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-openstack
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-openstack` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-openstack`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-openstack
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class’s name for the above group of nodes (the `cloudInstances.ClassReference` NodeGroup parameter). In the case of the OpenStack-based cloud provider, the instance class is the `OpenStackInstanceClass` custom resource that stores specific parameters of the machines.

> **i** An OpenStack cluster is deployed in a single region (`provider.region` in OpenStackClusterConfiguration). Zones from other regions cannot be used. See How do I create NodeGroups in availability zones? .

The module settings are set automatically based on the placement strategy chosen. In most cases, you do not have to configure the module manually.

If you need to configure a module because, say, you have a bare metal cluster and you need to enable additional instances from vSphere, then refer to the How to configure a Hybrid cluster in vSphere section.

> **i** **Note!** If the parameters provided below are changed, the **existing `Machines` are NOT redeployed** (new `Machines` will be created with the updated parameters). Redeployment is only performed when `NodeGroup` and `OpenStackInstanceClass` parameters are changed. You can learn more in the node-manager module’s documentation. To authenticate using the `user-authn` module, you need to create a new `Generic` application in the project’s Crowd.

List of required OpenStack services

A list of services required for Deckhouse Platform to work in OpenStack:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2.1
Network (Neutron)	v2.0
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to order a Load Balancer.

> **i** For Compute and Block Storage, the supported microversion depends on the installed OpenStack release. For Block Storage, microversion `3.0` is the minimum version and is used by default.

To ensure proper integration and functionality, the OpenStack user associated with the Deckhouse Platform should be granted the “member” role. This role assignment is essential to provide the necessary permissions for interacting with the listed services and managing resources within the OpenStack environment.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.additionalExternalNetworkNames**
    array of strings

    Specifies additional networks that can be connected to the VM. `cloud-controller-manager` uses them to insert `ExternalIP` to `.status.addresses` field in the Node API object.

    If you have instances in the cluster that use External Networks (other than those set out in the placement strategy), you must pass them via the `additionalExternalNetworkNames` parameter.

    Example:

    ```
    additionalExternalNetworkNames:
    - some-bgp-network
    ```

  - **settings.connection**
    object

    This section contains parameters required to connect to the cloud provider’s API.

    - **settings.connection.authURL**
      string

      An OpenStack Identity API URL.

    - **settings.connection.caCert**
      string

      Specify the CA x509 certificate used for signing if the OpenStack API has a self-signed certificate; The certificate must have a PEM format.

    - **settings.connection.domainName**
      string

      The domain name.

    - **settings.connection.password**
      string

      The user’s password.

    - **settings.connection.region**
      string

      The OpenStack region where the cluster will be deployed.

    - **settings.connection.tenantID**
      string

      The project ID. Cannot be used together with `tenantName`.

    - **settings.connection.tenantName**
      string

      The project name.

      Cannot be used together with `tenantID`.

    - **settings.connection.username**
      string

      The name of the user that has full project privileges.

- **settings.csiDriver**
  object

  Parameters for configuring the Cinder CSI Driver.

  - **settings.csiDriver.fsGroupPolicy**
    string

    Defines if the underlying volume supports changing ownership and permission of the volume before being mounted.

    - `ReadWriteOnceWithFSType`: Apply fsGroup only for ReadWriteOnce volumes (Kubernetes default behavior).
    - `File`: Always apply fsGroup. Use this for RWX volumes with Ceph backend.
    - `None`: Never change permissions.

    Default: `ReadWriteOnceWithFSType`

    Allowed values: `ReadWriteOnceWithFSType`, `File`, `None`

- **settings.externalNetworkNames**
  array of strings

  Additional networks that are connected to the VM. cloud-controller-manager uses them to insert ExternalIPs into `.status.addresses` in the Node API object.

  Example:

  ```
  externalNetworkNames:
  - KUBE-3
  - devops-internal
  ```

- **settings.ignoreVolumeMicroversion**
  boolean

  Setting for backwards compatibility. Enable if the cloud OpenStack version is less than 3.34 and you get error “Version 3.34 is not supported by the API. Minimum is 3.0 and maximum is 3.x” when ordering a PV. This will disable volumes online resize, but will restore ability to order new PVs. (original PR https://github.com/kubernetes/cloud-provider-openstack/pull/1986/)

  Default: `false`

- **settings.instances**
  object

  Instance parameters that are used when creating virtual machines.

  - **settings.instances.additionalNetworks**
    array of strings

    A list of networks to connect to the instance.

  - **settings.instances.imageName**
    string

    The name of the image.

  - **settings.instances.mainNetwork**
    string

    The path to the network that will serve as the primary network (the default gateway) for connecting to the VM.

  - **settings.instances.securityGroups**
    array of strings

    A list of securityGroups to assign to the provisioned instances. Defines firewall rules for the provisioned instances.

  - **settings.instances.sshKeyPairName**
    string

    The name of the OpenStack `keypair` resource; it is used for provisioning instances.

- **settings.internalNetworkNames**
  array of strings

  Additional networks that are connected to the VM. cloud-controller-manager uses them to insert InternalIPs into `.status.addresses` in the Node API object.

  Example:

  ```
  internalNetworkNames:
  - KUBE-3
  - devops-internal
  ```

- **settings.loadBalancer**
  object

Load Balancer parameters.

- **settings.loadBalancer.disabled**
  boolean

  Set to `true` to disable LoadBalancer support if the OpenStack cloud doesn't have Octavia (LBaaS) service available.

  Default: `false`

- **settings.loadBalancer.floatingNetworkID**
  string

  An ID of the external network for floating IPs.

- **settings.loadBalancer.subnetID**
  string

  An ID of the Neutron subnet to create the load balancer virtual IP in.

- **settings.podNetworkMode**
  string

  Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

  - `DirectRouting` — means that there is a direct routing between the nodes.
  - `DirectRoutingWithPortSecurityEnabled` — direct routing is enabled between the nodes, but only if the range of addresses of the internal network is explicitly allowed in OpenStack for Ports.
    - **Caution!** Make sure that the `username` can edit AllowedAddressPairs on Ports connected to the `internalNetworkName` network. Generally, an OpenStack user doesn't have such a privilege if the network has the `shared` flag set.
  - `VXLAN` — direct routing between the nodes isn't available; VXLAN should be used.

  Allowed values: `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`, `VXLAN`

- **settings.storageClass**
  object

  The module automatically creates StorageClasses that are available in OpenStack.

  - **settings.storageClass.default**
    Deprecated
    string

    The name of StorageClass that will be used in the cluster by default.

    If the parameter is omitted, the default StorageClass is either:

    - an arbitrary StorageClass present in the cluster that has the default annotation;
    - the first (in lexicographic order) StorageClass created by the module (in accordance with the order in OpenStack).

    > **Parameter is deprecated**. Instead, use the global parameter global.defaultClusterStorageClass.

    Example:

    ```
    default: ceph-ssd
    ```

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-hdd"
    - iscsi-fast
    ```

  - **settings.storageClass.topologyEnabled**
    boolean

    This feature enables driver to consider the topology constraints while creating the volume. It is used only during volume provisioning, existing PersistentVolumes are not affected.

    > **Warning.** If it is set to `false` all-new PersistentVolumes are provisioned without topology constraints.

    Default: `true`

- **settings.tags**
  object

  A dictionary of tags that will be available on all provisioned instances.

- **settings.zones**
  array of strings

  The default list of zones for provisioning instances. Can be redefined for each NodeGroup individually.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

OpenStackClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in OpenStack.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
- eu-3b
standard:
  internalNetworkDNSServers:
  - 8.8.8.8
  internalNetworkCIDR: 192.168.195.0/24
  internalNetworkSecurity: false
  externalNetworkName: external-network
provider:
  authURL: "<AUTH_URL>"
  domainName: "<DOMAIN_NAME>"
  tenantID: "<TENANT_ID>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  region: eu-3
masterNodeGroup:
  replicas: 1
  instanceClass:
    rootDiskSize: 50
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-3a: fast.eu-3a
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
    configDrive: false
    floatingIPPools:
    - public
    - shared
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - eu-3a
  - eu-3b
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**
  string

  Required value

  Allowed values: `OpenStackClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **masterNodeGroup**

  Required value

  The definition of the master’s NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

- **masterNodeGroup.instanceClass**
  object

  Required value

  Partial contents of the fields of the OpenStackInstanceClass.

  - **masterNodeGroup.instanceClass.additionalSecurityGroups**
    array of strings

    Security groups that will be applied to VM's network ports. Default group should also be added to this list.

    They allow you to set firewall rules for provisioned instances.

    The `SecurityGroups` may not be supported by the cloud provider.

  - **masterNodeGroup.instanceClass.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    additionalTags:
      project: cms-production
      severity: critical
    ```

  - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
    integer

    Etcd disk size in gigabytes.

    Default: `10`

    Example:

    ```
    etcdDiskSizeGb: 10
    ```

  - **masterNodeGroup.instanceClass.flavorName**
    string

    Required value

    Flavor of OpenStack servers.

    To get a list of all available flavors, run the command: `openstack flavor list`.

    For all non-master nodes it is advisable to use flavor's with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

    Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

  - **masterNodeGroup.instanceClass.imageName**
    string

    Required value

    Image to use while provisioning OpenStack servers.

    Use this command to get a list of available images: `openstack image list`.

    The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  - **masterNodeGroup.instanceClass.rootDiskSize**
    integer

    The size of a root disk (in gigabytes).

    This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.serverGroup**
  object

  Object groups instances together. The instances in the group are placed on the same hypervisor (affinity) or different hypervisors (anti-affinity). This allows you to increase the fault tolerance of the cluster.

  - **masterNodeGroup.serverGroup.manuallyManaged**
    object

        - **masterNodeGroup.serverGroup.manuallyManaged.id**
          string

          Required value

          The id of the existing `ServerGroup` object managed outside of Deckhouse.

      - **masterNodeGroup.serverGroup.policy**
        string

        Required value

        The policy that determines how instances are distributed among hypervisors.

        - `AntiAffinity` — instances are placed on different hypervisors. This policy is managed by Deckhouse.
        - `ManuallyManaged` — instances are placed on different hypervisors. This policy is managed outside of Deckhouse.

        Allowed values: `AntiAffinity`, `ManuallyManaged`

    - **masterNodeGroup.volumeTypeMap**
      object

      Required value

      A dictionary of disk types for storing etcd data and Kubernetes configuration files.

      Format of dictionary elements: `<AVAILABILITY ZONE>: <VOLUME TYPE>` (see the example).

      If the `rootDiskSize` parameter is specified, the same disk type will be used for the VM's boot drive.

      We recommend using the fastest disks provided by the provider in all cases.

      If the value specified in `replicas` exceeds the number of elements in the dictionary, the master nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if `replicas: 5`, then master-0, master-2, master-4 will have the `fast-eu-1a` disk type, while master-1, master-3 will have the `fast-eu-1b` disk type.

      Master nodes will be created in zones specified at this param following by alphabetical order. Based on the example below, `master-0` will be created in zone `eu-1a`, `master-1` will be created in zone `eu-1b`, and `master-2` will be created in zone `eu-1a`.

      Useful commands:

      - `openstack availability zone list` — get list of availability zones.
      - `openstack volume type list` — get list of volume types.

      Example:

      ```
      volumeTypeMap:
        eu-1a: fast-eu-1a
        eu-1b: fast-eu-1b
      ```

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating CloudPermanent nodes (for example, dedicated front nodes or gateways).

  For CloudEphemeral nodes, use a separate NodeGroup resource and an OpenStackInstanceClass.

  - **nodeGroups.instanceClass**
    object

    Partial contents of the fields of the OpenStackInstanceClass.

    - **nodeGroups.instanceClass.additionalNetworks**
      array of strings

      Paths to networks that VirtualMachines secondary NICs will connect to. To get a list of all available networks, run the command: `openstack network list`.

      Example:

      ```
      additionalNetworks:
      - BGP-network-VLAN-3894
      - External-VLAN-3699
      ```

    - **nodeGroups.instanceClass.additionalSecurityGroups**
      array of strings

      Security groups that will be applied to VM's network ports. Default group should also be added to this list.

      They allow you to set firewall rules for provisioned instances.

      The `SecurityGroups` may not be supported by the cloud provider.

    - **nodeGroups.instanceClass.additionalTags**
      object

The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

Example:

```
additionalTags:
  project: cms-production
  severity: critical
```

- **nodeGroups.instanceClass.configDrive**
  boolean

  Not required value.

  Specifies whether an additional disk containing the bootstrapping configuration will be mounted to the node.

  You must set it if DHCP is disabled in the `mainNetwork`.

  Default: `false`

- **nodeGroups.instanceClass.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  To get a list of all available flavors, run the command: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor's with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

  Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

- **nodeGroups.instanceClass.floatingIPPools**
  array of strings

  A list of networks to assign Floating IPs to nodes.

- **nodeGroups.instanceClass.imageName**
  string

  Required value

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

- **nodeGroups.instanceClass.mainNetwork**
  string

  Required value

  Path to the network that VirtualMachines primary NICs will connect to (default gateway). To get a list of all available networks, run the command: `openstack network list`.

- **nodeGroups.instanceClass.networksWithSecurityDisabled**
  array of strings

  A list of `mainNetwork` and `additionalNetworks` in which `SecurityGroups` and `AllowedAddressPairs` on ports **cannot be configured**.

- **nodeGroups.instanceClass.rootDiskSize**
  integer

  The size of a root disk (in gigabytes).

  This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

- **nodeGroups.name**
  string

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**
  object

  A list of labels to attach to cluster resources.

  The same as the `metadata.labels` standard field.

  Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

  Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  The number of nodes to create.

- **nodeGroups.volumeTypeMap**
  object

  A dictionary of disk types for root drive.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <VOLUME TYPE>` (see the example).

  If the value specified in `replicas` exceeds the number of elements in the dictionary, the nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if `replicas: 5`, then worker-0, worker-2, worker-4 will have the `fast-eu-1a` disk type, while worker-1, worker-3 will have the `fast-eu-1b` disk type.

  > **i** **Caution.** Limits by availability zones works only when using the `zones` parameter (`nodeGroups.zones`). If `zones` parameter is not specified, all availability zones will be used.

  Useful commands:

  - `openstack availability zone list` — get list of availability zones.
  - `openstack volume type list` — get list of volume types.

  Example:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups.zones**
  array of strings

  Not required value.

  A limited set of zones in which nodes can be created.

  The zones must belong to the `provider.region` region. If the `zones` parameter is set, only zones from that

parameter can be used.

- **provider**
  object

  Required value

  Contains settings to connect to the OpenStack API.

  These settings are the same as those in the `connection` field of the cloud-provider-openstack module.

  - **provider.authURL**
    string

    An OpenStack Identity API URL.

  - **provider.caCert**
    string

    Specify the CA x509 certificate used for signing if the OpenStack API has a self-signed certificate. Certificate should be passed in PEM format as multiline string.

    Example:

    ```
    caCert: |
      -----BEGIN CERTIFICATE-----
      MIIFyDCCBLCgAwIBAgIQBwDIWH1asdaKNaALUa4NUzANBgkqhkiG9w0BAQsFADBc
      ...
      -----END CERTIFICATE-----
    ```

  - **provider.domainName**
    string

    The domain name.

    `OS_USER_DOMAIN_NAME` variable from the openrc file.

  - **provider.password**
    string

    The user’s password.

  - **provider.region**
    string

    The OpenStack region where the cluster will be deployed.

    The cluster runs in a single region. Master nodes, as well as CloudPermanent and CloudEphemeral nodes, can be created only in zones of this region.

  - **provider.tenantID**
    string

    The project id.

    Cannot be used together with `tenantName`.

  - **provider.tenantName**
    string

    The project name.

    Cannot be used together with `tenantID`.

  - **provider.username**
    string

    The name of the user that has full project privileges.

- **simple**
  object

  Settings for the `Simple` layout.

  - **simple.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **simple.externalNetworkName**
    string

    Required value

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **simple.podNetworkMode**
    string

Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

- `DirectRouting` — nodes are directly routed (SecurityGroups are disabled in this mode).
- `VXLAN` — direct routing does NOT work between nodes, VXLAN must be used (SecurityGroups are disabled in this mode).

> **Caution.** After changing this parameter, you need to run `dhctl converge` command for the changes to take effect.

> **Caution.** All cluster nodes must be rebooted after switching work mode from/to VXLAN.

Default: `VXLAN`

Allowed values: `VXLAN`, `DirectRouting`

- **simpleWithInternalNetwork**
  object

  Settings for the `SimpleWithInternalNetwork` layout.

  - **simpleWithInternalNetwork.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **simpleWithInternalNetwork.externalNetworkName**
    string

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **simpleWithInternalNetwork.internalSubnetID**
    string

    The UUID of the subnet in which the cluster nodes will run.

    Set either this parameter or `internalSubnetName`.

    Use the UUID when the subnet has no meaningful name. For example, in Selectel the subnet name is generated automatically, and in the web interface of some providers it is not shown at all. The UUID is visible in the `Subnets` column of the `openstack network list` output and in the `ID` column of the `openstack subnet list` output.

    Pattern: `^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$`

  - **simpleWithInternalNetwork.internalSubnetName**
    string

    The name of the subnet in which the cluster nodes will run.

    Set either this parameter or `internalSubnetID`.

    Specify a subnet, not a network. A network can have several subnets, and the subnet name does not have to match the network name. The `openstack subnet list -c ID -c Name -c Network` command lists the subnets with their names and networks.

  - **simpleWithInternalNetwork.masterWithExternalFloatingIP**
    boolean

    Defines if Floating IP must be assigned to master nodes.

    Default: `true`

  - **simpleWithInternalNetwork.podNetworkMode**
    string

    Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

    - `DirectRouting` — nodes are directly routed (SecurityGroups are disabled in this mode).
    - `DirectRoutingWithPortSecurityEnabled` — direct routing is enabled between the nodes, but only if the range of addresses of the internal network is explicitly allowed in OpenStack for Ports:
      - **Caution.** Make sure that the `username` can edit `AllowedAddressPairs` on Ports connected to the `internalNetworkName` network. Usually, an OpenStack user doesn't have such a privilege if the network has the `shared` flag set.
    - `VXLAN` — direct routing does NOT work between nodes, VXLAN must be used (`SecurityGroups` are disabled in this mode).

    > **Caution.** After changing this parameter, you need to run `dhctl converge` command for the changes to take effect.

> **Caution.** All cluster nodes must be rebooted after switching work mode from/to VXLAN.

Default: `DirectRoutingWithPortSecurityEnabled`

Allowed values: `VXLAN`, `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`

- **sshAllowList**
  array of strings

  A list of CIDR's allowed to connect to nodes via SSH.

  By default, `0.0.0.0/0`.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.bastion**
    object

    The definition of the bastion instance.

    - **standard.bastion.instanceClass**
      object

      Partial contents of the fields of the OpenStackInstanceClass.

      - **standard.bastion.instanceClass.additionalTags**
        object

        The additional tags to attach to the instance created (in addition to those specified in the cloud provider configuration).

        Example:

        ```
        additionalTags:
          project: cms-production
          severity: critical
        ```

      - **standard.bastion.instanceClass.flavorName**
        string

        Required value

        Flavor of OpenStack servers.

        To get a list of all available flavors, run the command: `openstack flavor list`.

      - **standard.bastion.instanceClass.imageName**
        string

        Required value

        Image to use while provisioning OpenStack servers.

        Use this command to get a list of available images: `openstack image list`.

        The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

      - **standard.bastion.instanceClass.rootDiskSize**
        integer

        The size of a root disk (in gigabytes).

        This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

        Default: `50`

    - **standard.bastion.volumeType**
      string

      Root disk type.

    - **standard.bastion.zone**
      string

      The zone to create an instance for the bastion node.

  - **standard.externalNetworkName**

string

Required value

The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

- **standard.internalNetworkCIDR**
  string

  Required value

  Routing for the internal cluster network.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standard.internalNetworkDNSServers**
  array of strings

  Required value

  A list of addresses of the recursive DNSs of the internal cluster network.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **standard.internalNetworkSecurity**
  boolean

  Defines whether SecurityGroups and AllowedAddressPairs must be configured for ports of the internal network.

  When enabled, DP creates a security group named after the cluster `prefix` with the following default inbound rules:

  - TCP/22 (SSH) from CIDRs in `sshAllowList` (default `0.0.0.0/0`)
  - ICMP from `0.0.0.0/0`
  - TCP NodePorts `30000–32767` from `0.0.0.0/0`
  - all inbound traffic from members of the same security group

  For details, see OpenStack: Layouts.

  Default: `true`

- **standardWithNoRouter**
  object

  Settings for the `StandardWithNoRouter` layout.

  - **standardWithNoRouter.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **standardWithNoRouter.externalNetworkName**
    string

    Required value

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **standardWithNoRouter.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster network.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **standardWithNoRouter.internalNetworkSecurity**
    boolean

    Defines whether SecurityGroups and AllowedAddressPairs must be configured for ports of the internal network.

    When enabled, DP creates a security group named after the cluster `prefix` with the following default inbound rules:

    - TCP/22 (SSH) from CIDRs in `sshAllowList` (default `0.0.0.0/0`)
    - ICMP from `0.0.0.0/0`
    - TCP NodePorts `30000–32767` from `0.0.0.0/0`
    - all inbound traffic from members of the same security group

    For details, see OpenStack: Layouts.

    Default: `true`

- **tags**
  object

  Not required value.

  A dictionary of tags to create on all resources that support this feature.

  You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **zones**
  array of strings

  Not required value.

  The globally restricted set of availability zones that this cloud provider works with.

  Specify only zones of the `provider.region` region. The cluster is deployed in a single OpenStack region — zones from other regions cannot be used.

  To get the list of zones in the region, run: `openstack availability zone list --compute`.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> The module lifecycle stage: General Availability

> To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.

> The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:
>
> - `cloud-config.service`;
> - `cloud-final.service`;
> - `cloud-init.service`.

> The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To manage resources in an OpenStack cloud, Deckhouse connects to the OpenStack API.
The list of OpenStack API services that need to be accessed for deployment is available in the settings section.
The user credentials required to connect to the OpenStack API are located in the openrc file (OpenStack RC file).

You can read the OpenStack documentation to get information about getting and using an openrc file with the standard OpenStack web interface.

The interface for getting an openrc file may differ if you use an OpenStack API of a cloud provider.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

OpenStackInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of OpenStack servers used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines' secondary NICs will connect to.

    By default: the value from `OpenStackCloudDiscoveryData` is used.

    Example:

    ```
    additionalNetworks:
    - BGP-network-VLAN-3894
    - External-VLAN-3699
    ```

  - **spec.additionalSecurityGroups**
    array of strings

    Security groups that will be applied to VM's network ports. Default group should also be added to this list.

    They allow you to set firewall rules for provisioned instances.

    The `SecurityGroups` may not be supported by the cloud provider.

    Example:

    ```
    additionalSecurityGroups:
    - security-group-1
    - security-group-2
    ```

  - **spec.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    additionalTags:
      project: cms-production
      severity: critical
    ```

  - **spec.capacity**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

    Instance capacity for *non-standard* instance types (the `flavorName` parameter).

    Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

    The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`m1.medium`, `m1.large`, etc.) is embedded in Deckhouse.

    - **spec.capacity.cpu**
      integer or string

      Required value

      Node vCPU resources.

      Pattern: `^[0-9]+m?$`

      Example:

      ```
      cpu: 1000m
      ```

    - **spec.capacity.memory**
      integer or string

Required value

Node memory resources.

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Example:

```
memory: 1500Mi
```

- **spec.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  Get a list of all available flavors: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor's with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

  Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

  Example:

  ```
  flavorName: m1.medium
  ```

- **spec.imageName**
  string

  Not required value.

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  By default: either `OpenStackCloudDiscoveryData` value or `instances.imageName` is used.

  Example:

  ```
  imageName: ubuntu-18-04-cloud-amd64
  ```

- **spec.mainNetwork**
  string

  Not required value.

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  By default: the value from `OpenStackCloudDiscoveryData` is used.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **spec.preemptible**
  boolean

  Order preemptible virtual machines.

  The cloud can interrupt such a machine at any time.

  The parameter applies in Selectel and only to CloudEphemeral node groups managed by Cluster API (CAPI). In another OpenStack cloud, or when the node group is managed by Machine Controller Manager (MCM), the platform creates regular virtual machines and fires the `OpenStackPreemptibleUnsupported` alert.

  Default: `false`

  Example:

  ```
  preemptible: true
  ```

- **spec.rootDiskSize**
  integer

  Not required value.

  The size of a root disk (in gigabytes). This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

  Example:

```
rootDiskSize: 20
```

- **spec.volumeTypeMap**
  object

  Root disk volume types in different availability zones.

  Dictionary elements use the `<AVAILABILITY_ZONE>: <VOLUME_TYPE>` format.

  Warning. Changing the parameter recreates the nodes of every group that references this OpenStackInstanceClass.

  Set this parameter if volume types in your cloud are bound to availability zones. In such a cloud, Cinder creates the root disk with the default volume type, which is bound to a single zone, so provisioning a node in another zone fails with the `HTTP 400 Availability zone '…' is invalid` error.

  For a zone listed in the dictionary, the platform creates the root disk of the specified type in the same zone where the node is provisioned. For the other zones, Cinder picks the volume type.

  The parameter takes effect only together with `rootDiskSize`, because without it the root disk is local and Cinder is not involved. The parameter only works in node groups running on the CAPI engine. The MCM engine ignores it.

  The `openstack availability zone list` command lists availability zones. The `openstack volume type list` command lists volume types.

  Example:

  ```
  volumeTypeMap:
    ru-3a: fast.ru-3a
    ru-3b: fast.ru-3b
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of OpenStack servers used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines’ secondary NICs will connect to.

    By default: the value from `OpenStackCloudDiscoveryData` is used.

    Example:

    ```
    additionalNetworks:
    - BGP-network-VLAN-3894
    - External-VLAN-3699
    ```

  - **spec.additionalSecurityGroups**
    array of strings

    Security groups that will be applied to VM’s network ports. Default group should also be added to this list.

    They allow you to set firewall rules for provisioned instances.

    The `SecurityGroups` may not be supported by the cloud provider.

    Example:

    ```
    additionalSecurityGroups:
    - security-group-1
    - security-group-2
    ```

  - **spec.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    additionalTags:
      project: cms-production
      severity: critical
    ```

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `flavorName` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`m1.medium`, `m1.large`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  Get a list of all available flavors: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor's with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

  Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

  Example:

  ```
  flavorName: m1.medium
  ```

- **spec.imageName**
  string

  Not required value.

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  By default: either `OpenStackCloudDiscoveryData` value or `instances.imageName` is used.

  Example:

  ```
  imageName: ubuntu-18-04-cloud-amd64
  ```

- **spec.mainNetwork**
  string

  Not required value.

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  By default: the value from `OpenStackCloudDiscoveryData` is used.

  Example:

```
mainNetwork: k8s-msk-178
```

- **spec.preemptible**
  boolean

  Order preemptible virtual machines.

  The cloud can interrupt such a machine at any time.

  The parameter applies in Selectel and only to CloudEphemeral node groups managed by Cluster API (CAPI). In another OpenStack cloud, or when the node group is managed by Machine Controller Manager (MCM), the platform creates regular virtual machines and fires the `OpenStackPreemptibleUnsupported` alert.

  Default: `false`

  Example:

  ```
  preemptible: true
  ```

- **spec.rootDiskSize**
  integer

  Not required value.

  The size of a root disk (in gigabytes). This parameter also has influence on type of volume that will be used for root disk; the “How to use rootDiskSize and when it is preferred” section describes how to use it.

  Example:

  ```
  rootDiskSize: 20
  ```

- **spec.volumeTypeMap**
  object

  Root disk volume types in different availability zones.

  Dictionary elements use the `<AVAILABILITY_ZONE>: <VOLUME_TYPE>` format.

  Warning. Changing the parameter recreates the nodes of every group that references this OpenStackInstanceClass.

  Set this parameter if volume types in your cloud are bound to availability zones. In such a cloud, Cinder creates the root disk with the default volume type, which is bound to a single zone, so provisioning a node in another zone fails with the `HTTP 400 Availability zone '…' is invalid` error.

  For a zone listed in the dictionary, the platform creates the root disk of the specified type in the same zone where the node is provisioned. For the other zones, Cinder picks the volume type.

  The parameter takes effect only together with `rootDiskSize`, because without it the root disk is local and Cinder is not involved. The parameter only works in node groups running on the CAPI engine. The MCM engine ignores it.

  The `openstack availability zone list` command lists availability zones. The `openstack volume type list` command lists volume types.

  Example:

  ```
  volumeTypeMap:
    ru-3a: fast.ru-3a
    ru-3b: fast.ru-3b
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How do I set up LoadBalancer?

> ⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

The `annotations` field of the corresponding inlet configuration in the IngressNginxController resource supports the following annotations:

- `loadbalancer.openstack.org/node-selector`: Selects the nodes that will be used as LoadBalancer targets.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id`: Instructs OpenStack CCM to use a pre-created Octavia load balancer.
- `loadbalancer.openstack.deckhouse.io/load-balancer-address`: Instructs OpenStack CCM to associate a pre-allocated floating IP with the load balancer it creates.

DP automatically adds the specified annotations to the generated Service object of type LoadBalancer.

When using the `loadbalancer.openstack.deckhouse.io/load-balancer-id` annotation, the load balancer must meet the following requirements:

- reside in the cluster subnet
- be in the `ACTIVE` state

If `loadbalancer.openstack.deckhouse.io/load-balancer-id` is used to reference a pre-created load balancer with a custom name, associate the floating IP with its VIP port before creating the cluster. In this case, do not specify the `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotation.

When using only `loadbalancer.openstack.deckhouse.io/load-balancer-address`, the floating IP must meet the following requirements:

- not be associated with any port
- reside in the floating network configured for OpenStack CCM

If the specified floating IP is unavailable, OpenStack CCM will not be able to assign an external IP address to the Service object.

Do not add the `loadbalancer.openstack.deckhouse.io/load-balancer-id` or `loadbalancer.openstack.deckhouse.io/load-balancer-address` annotations to application Ingress resources. Specify them only in the IngressNginxController configuration. DP will add them to the generated Service object, which is processed by `openstack-cloud-controller-manager`.

IngressNginxController with a pre-created load balancer

In the example below:

- Ingress controller pods are scheduled on frontend nodes.
- `loadbalancer.openstack.org/node-selector` limits the load balancer target pool to those frontend nodes.
- `loadbalancer.openstack.deckhouse.io/load-balancer-id` references a pre-created Octavia load balancer whose VIP port already has a floating IP associated with it.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-id: "df7c6f73-8c68-4a11-a3e2-6268a655ce9b"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

IngressNginxController with a pre-allocated floating IP

In the example below, OpenStack CCM creates a load balancer and associates the specified unassigned floating IP with it:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.deckhouse.io/load-balancer-address: "203.0.113.10"
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

How do I create NodeGroups in availability zones?

An OpenStack cluster is deployed in a single region, which is set by the `provider.region` parameter of the OpenStackClusterConfiguration resource. Nodes can be created only in availability zones of this region. Using zones from other regions is not supported.

To get the list of availability zones in the region, run:

```
openstack --os-region-name <REGION> availability zone list --compute
```

CloudPermanent node groups are defined in the `nodeGroups` section of the OpenStackClusterConfiguration resource. To limit availability zones, use the `nodeGroups[].zones` parameter.

CloudEphemeral nodes are created with a separate NodeGroup resource with `nodeType: CloudEphemeral`. Availability zones for them are set in the `spec.cloudInstances.zones` parameter. If the `zones` parameter is specified in OpenStackClusterConfiguration, the required zones must also be added to it.

> **i** If the admission webhook returns the `unknown zone` error when creating a NodeGroup, make sure that the specified zone belongs to the `provider.region` region and, when using the `zones` parameter, is included in the OpenStackClusterConfiguration zones list.

Nodes from another region can be added to the cluster only manually as Static nodes. The `cloud-provider-openstack` module does not create such nodes.

> **⚠** After changing OpenStackClusterConfiguration, run the `dhctl converge` command. For details, see Adding and managing cloud nodes.

Example NodeGroup for CloudEphemeral nodes:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: workers
    minPerZone: 1
    maxPerZone: 1
    zones:
    - eu-3a
    - eu-3b
```

If the instance class of the group sets the `rootDiskSize` parameter, the root disk is created in Cinder. In clouds where volume types are bound to availability zones, set the volume type for each zone in the `volumeTypeMap` parameter of the OpenStackInstanceClass resource. Otherwise, provisioning a node in a zone where the default volume type is unavailable fails with the `HTTP 400 Availability zone '…' is invalid` error.

How do I create a hybrid cluster?

A hybrid cluster combines bare metal and OpenStack nodes. To create such a cluster, you need an L2 network between all nodes of the cluster.

> **i** The Deckhouse Platform allows to set a prefix for the names of CloudEphemeral nodes added to a hybrid cluster with Static master nodes. To do this, use the `instancePrefix` parameter of the `node-manager` module. The prefix specified in the parameter will be added to the name of all CloudEphemeral nodes added to the cluster. It is not possible to set a prefix for a specific NodeGroup.

To set up a hybrid cluster, follow these steps:

1. Delete flannel from kube-system: `d8 k -n kube-system delete ds flannel-ds`.
2. Enable and configure the module.
3. Create one or more OpenStackInstanceClass custom resources.
4. Create one or more NodeManager custom resources for specifying the number of machines and managing the provisioning process in the cloud.

> **i** **Caution!** Cloud-controller-manager synchronizes OpenStack and Kubernetes states by deleting Kubernetes nodes that are not in OpenStack. In a hybrid cluster, such behavior does not always make sense. That is why cloud-controller-manager automatically skips Kubernetes nodes that do not have the `--cloud-provider=external` parameter (Deckhouse inserts `static://` into nodes in `.spec.providerID`, and cloud-controller-manager ignores them).

Attaching storage devices to instances in a hybrid cluster

To use PersistentVolumes on OpenStack nodes, you must create StorageClass with the appropriate OpenStack volume type. The `openstack volume type list` command lists all available types.

Here is the example config for the `ceph-ssd` volume type:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ceph-ssd
provisioner: csi-cinderplugin # have to be like this
parameters:
  type: ceph-ssd
volumeBindingMode: WaitForFirstConsumer
```

How do I create an image in OpenStack?

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an OpenStack RC (openrc) file containing credentials for accessing the OpenStack API:

   > **i** The interface for getting an openrc file may differ depending on the OpenStack provider. If the provider has a standard interface for OpenStack, you can download the openrc file using the following instruction.

3. Otherwise, install the OpenStack client using this instruction.

   Also, you can run the container and mount an openrc file and a downloaded Ubuntu image in it:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize the environment variables from the openrc file:

   ```
   source /openrc
   ```

5. Get a list of available disk types:

```
/ # openstack volume type list
+--------------------------------------+--------------+-----------+
| ID                                   | Name         | Is Public |
+--------------------------------------+--------------+-----------+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
| 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
+--------------------------------------+--------------+-----------+
```

6. Create an image, pass the disk format to use (if OpenStack does not support local disks or these disks don’t fit):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Check that the image was created successfully:

```
/ # openstack image show ubuntu-18-04-cloud-amd64
+------------------+------------------------------------
```

This behavior is especially important for clusters where `cinder-csi-plugin` pods run for a long time without restart, while the OpenStack service catalog may change between the initial authentication and subsequent API requests.

How to check whether the provider supports SecurityGroups?

Run the following command: `openstack security group list`. If there are no errors in the output, then Security Groups are supported.

How to set up online disk resize

The OpenStack API states that the resize is completed successfully. However, Nova does not get any information about the resize from Cinder. As a result, the size of the disk in the guest OS remains the same.

To get rid of this problem, you need to insert the Nova API access parameters into the `cinder.conf` file, e.g., as follows:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Source...

How to use `rootDiskSize` and when it is preferred?

Disks in OpenStack

The node disk can be local or network. A local disk in OpenStack, is an ephemeral disk, and a network disk is a persistent disk (cinder storage). Nodes with local disks cannot migrate between hypervisors.

- A network disk is preferred for the master node so that the node can migrate between hypervisors.
- A local disk is preffered for the ephemeral node to save on cost. Not all cloud providers support the use of local disks. If local disks are not supported, you have to use network disks for ephemeral nodes.

Local disk (ephemeral)	Network disk (persistent)
Cheaper	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

The `rootDiskSize` parameter

The `OpenStackInstanceClass` has a `rootDiskSize` parameter, and OpenStack flavor has a disk size parameter. Which disk will be ordered depending on the combination of parameters is shown in the table:

	flavor disk size = 0	**flavor disk size > 0**
`rootDiskSize` is not specified	*You need to set the size*. Without specifying the size, there will be an error creating a VM.	Local disk with size according to the flavor
`rootDiskSize` is specified	Network disk with the `rootDiskSize` size	Network disk (rootDiskSize) and local disk (according to the flavor). Avoid using this option, as the cloud provider will charge for both disks.

> **i** Please note, that to create a node with the `CloudEphemeral` type in a zone other than zone A, you must first create a flavor with a disk of the required size. The rootDiskSize parameter does not need to be specified.

Network disk is recommended for master nodes and bastion host

- Use flavor with a zero disk size.
- Set the `rootDiskSize` in the `OpenStackInstanceClass`.
- Check the disk type. The disk type will be taken from the OS image if it is set. If it is not set, the disk type will be taken from volumeTypeMap.

Local disk is recommended for ephemeral nodes

- Use flavor with the specified disk size.

- Do not use the `rootDiskSize` parameter in the `OpenStackInstanceClass`.
- Check the disk type. The disk type will be taken from the OS image if it is set. If it is not set, the default disk type of the cloud provider will be used.

How do I check the disk volume in a flavor?

```
# openstack flavor show m1.medium-50g -c disk
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

How to override a default volume type of cloud provider?

If there are several types of disks in a *cloud provider*, you can set a default disk type for the image in order to select a specific VM's disk type. To do this, specify the name of a disk type in the image metadata.

Also, you may need to create a custom OpenStack image; the "How do I create an image in OpenStack" section describes how to do it

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

OFFLINE disk resize

Some cloud providers may not support ONLINE disk resizing. If you get the following error, then you need to reduce the number of StatefulSet replicas to 0, wait for disk resizing and return the number of replicas that was before the start of the operation.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

How to create preemptible nodes in Selectel?

Selectel triggers preemption on the presence of the raw Nova tag `preemptible`. Set the `preemptible` parameter of the OpenStackInstanceClass to `true` — the tag is then attached to the instance automatically.

Example:

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: worker-preempt
spec:
  flavorName: SL1.4-8192
  imageName: Ubuntu 24.04 LTS 64-bit
  rootDiskSize: 30
  preemptible: true
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-preempt
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: worker-preempt
    minPerZone: 2
    maxPerZone: 4
    zones: [ru-3a]
```

Limitations:

- The `preemptible` tag is a Selectel-specific mechanism, and the CAPI provider template emits it only when the cluster's `connection.authURL` points at Selectel (`selcloud.ru` / `selectel`). On other OpenStack providers the tag is silently dropped: the node is still created, but as a regular (non-preemptible) VM.
- Supported only for CloudEphemeral NodeGroups running on the CAPI engine. If an OpenStackInstanceClass with

`preemptible: true` is used by a NodeGroup running on the MCM engine, the field is silently ignored — no MCM MachineClass field maps to a raw Nova tag. Nodes are created as regular VMs.
- Both situations above raise the **OpenStackPreemptibleUnsupported** Prometheus alert (with `reason=mcm` or `reason=non-selectel`) so the operator learns about the silent drop instead of discovering it weeks later from an unchanged billing line. The alert description carries the exact commands to fix each case.
- The management engine is selected per NodeGroup. If the same OpenStackInstanceClass with `preemptible: true` is used simultaneously by NodeGroups running on CAPI and MCM, the MCM NodeGroup keeps firing the alert — use separate OpenStackInstanceClass resources for different engines when the preemptibility contract differs.

⚠ Changing the `preemptible` parameter in an existing OpenStackInstanceClass changes the OpenStackMachineTemplate and causes all nodes in the corresponding NodeGroup to be recreated.

If a preemptible instance is terminated without a `graceful shutdown`, the corresponding node transitions to the `NotReady` state. After the missing instance is detected, CAPO and MachineHealthCheck initiate the creation of a new Machine and virtual machine. Until recovery is complete, pods on the lost node may still appear in the Kubernetes API with the `Running` status even though they are no longer actually available.

What to do if switching to nodes in lower-priority groups takes a long time?

If switching to nodes in lower-priority groups takes a long time, follow the instructions.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

i The module lifecycle stage: General Availability

Below are two simple examples of OpenStack cloud provider configuration.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

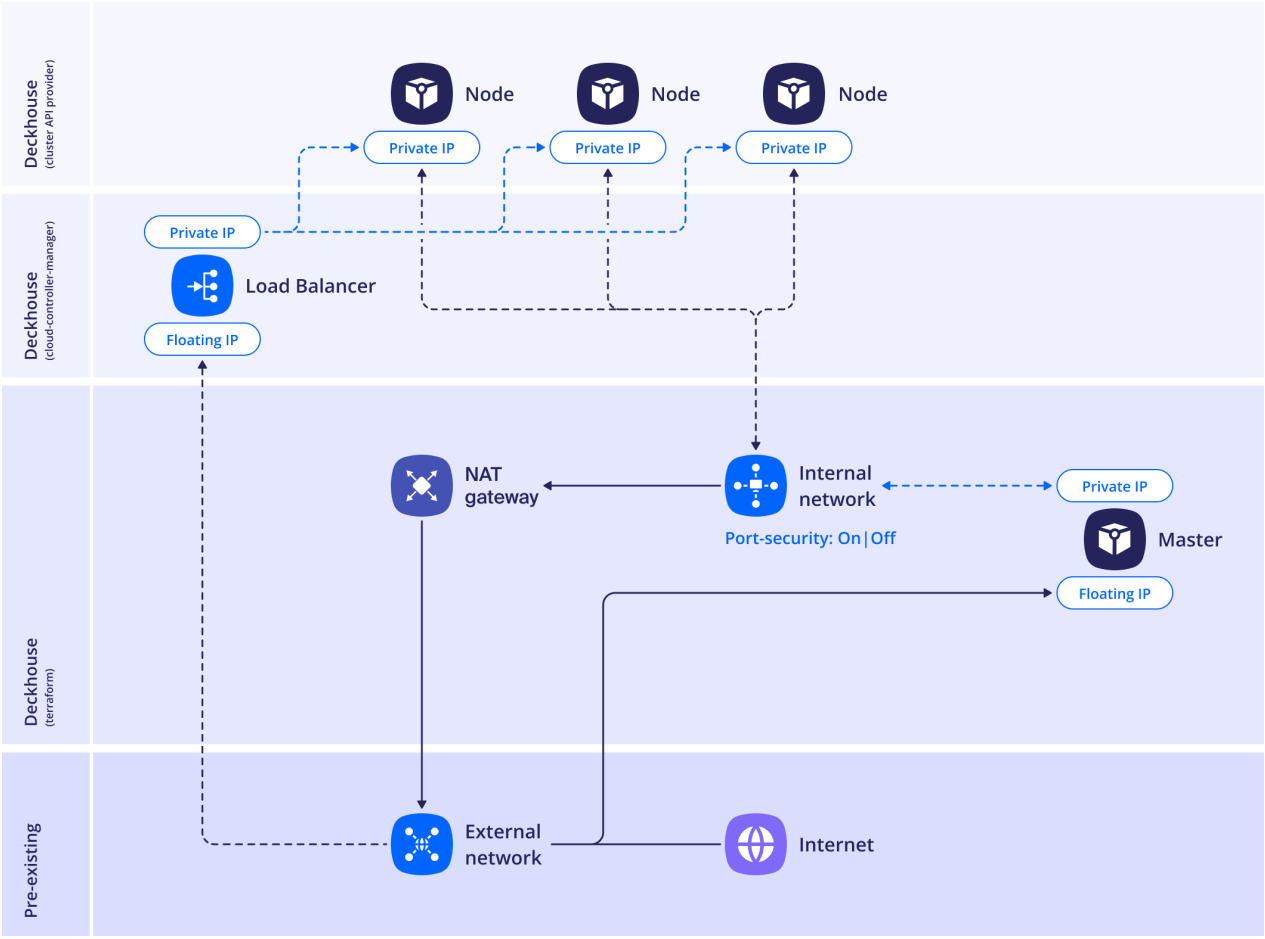
Cloud provider - OpenStack: Layouts

The module lifecycle stage: General Availability

Four layouts are supported. Below is more information about each of them.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. Note that the floating IP is assigned to the master node.



> **Caution!** If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port 6443. To avoid this, we recommend using the SimpleWithInternalNetwork placement strategy or Standard strategy with bastion host.

The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. The module creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- allow incoming traffic over the TCP protocol on port `22` from CIDRs listed in `sshAllowList` (default `0.0.0.0/0`)
- allow incoming traffic over the ICMP protocol from `0.0.0.0/0`
- allow incoming traffic over the TCP protocol on ports `30000-32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default
- allow any incoming traffic from nodes in the same security group

In addition to the group created by the module, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- for master nodes — in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, default true.
  externalNetworkName: shared               # Required.
  bastion:
    zone: ru2-b                           # Optional.
    volumeType: fast-ru-2b                  # Optional.
    instanceClass:
      flavorName: m1.large                  # Required.
```

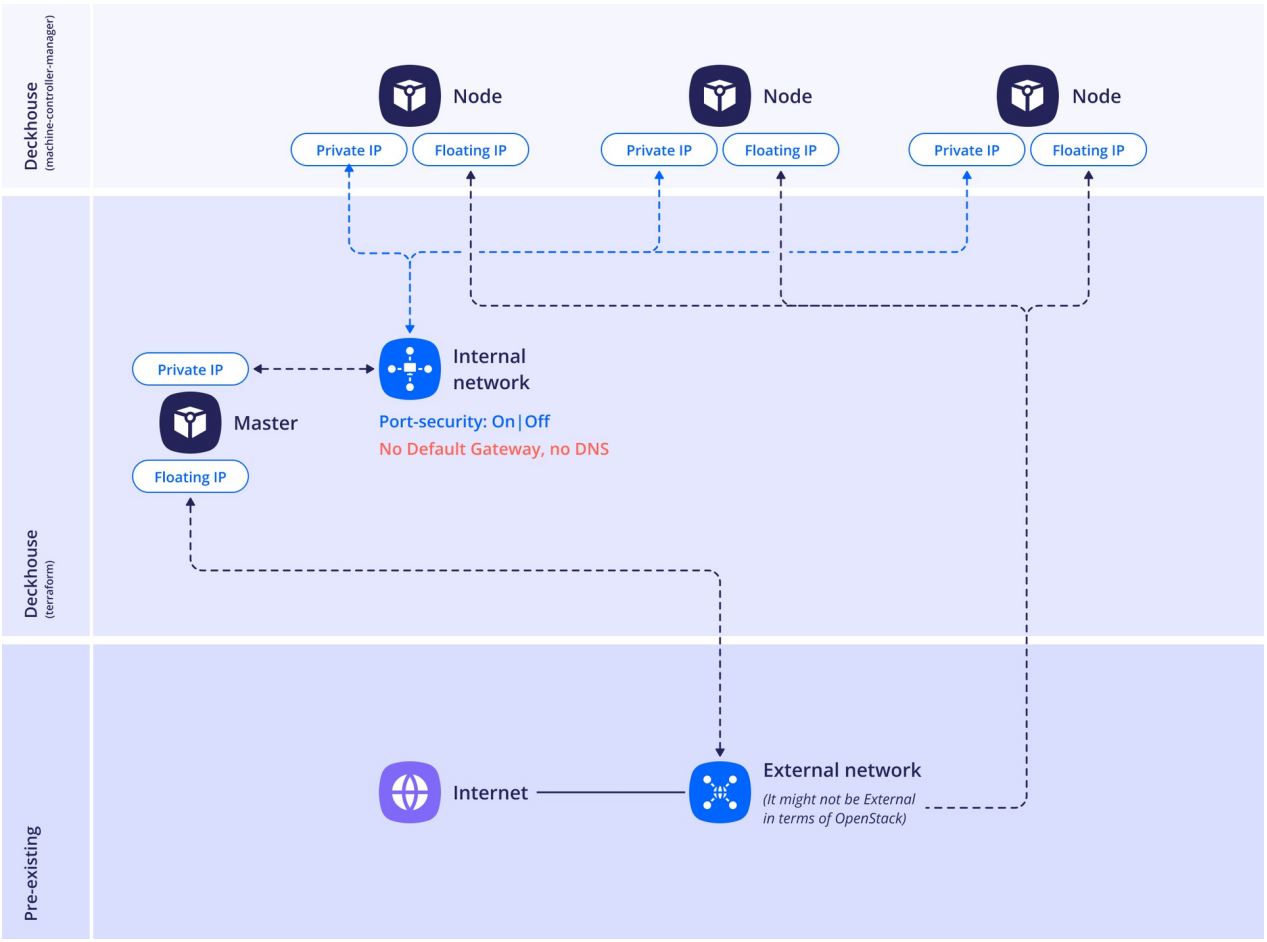
```
    imageName: ubuntu-20-04-cloud-amd64     # Required.
    rootDiskSize: 50                      # Optional, default 50 gigabytes.
    additionalTags:
      severity: critical                  # Optional.
      environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                   # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive is
    # required during vm bootstrap process. It's needed if there
    # is no dhcp in network that is used as default gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                    # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

Caution! This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a load balancer with the Floating IP. An internal load balancer with the virtual IP in the public network is only accessible to cluster nodes.

Caution! In this strategy, it is necessary to explicitly specify the name of the internal network in `additionalNetworks` when creating an `OpenStackInstanceClass` in the cluster.



The `internalNetworkSecurity` parameter (default `true`) enables the creation of a security group (SecurityGroup) when a cluster is created. The module creates the group named after the cluster prefix (`prefix`) and assigns it to the nodes.

The following inbound rules will be created:

- allow incoming traffic over the TCP protocol on port `22` from CIDRs listed in `sshAllowList` (default `0.0.0.0/0`)
- allow incoming traffic over the ICMP protocol from `0.0.0.0/0`
- allow incoming traffic over the TCP protocol on ports `30000`–`32767` for services of the `NodePort` type. Inbound UDP traffic to `NodePort` ports is not allowed by default
- allow any incoming traffic from nodes in the same security group

In addition to the group created by the module, you can attach your own security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and specify them in `additionalSecurityGroups`:

- for master nodes — in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource
- for static nodes — in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource
- for ephemeral nodes — in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource

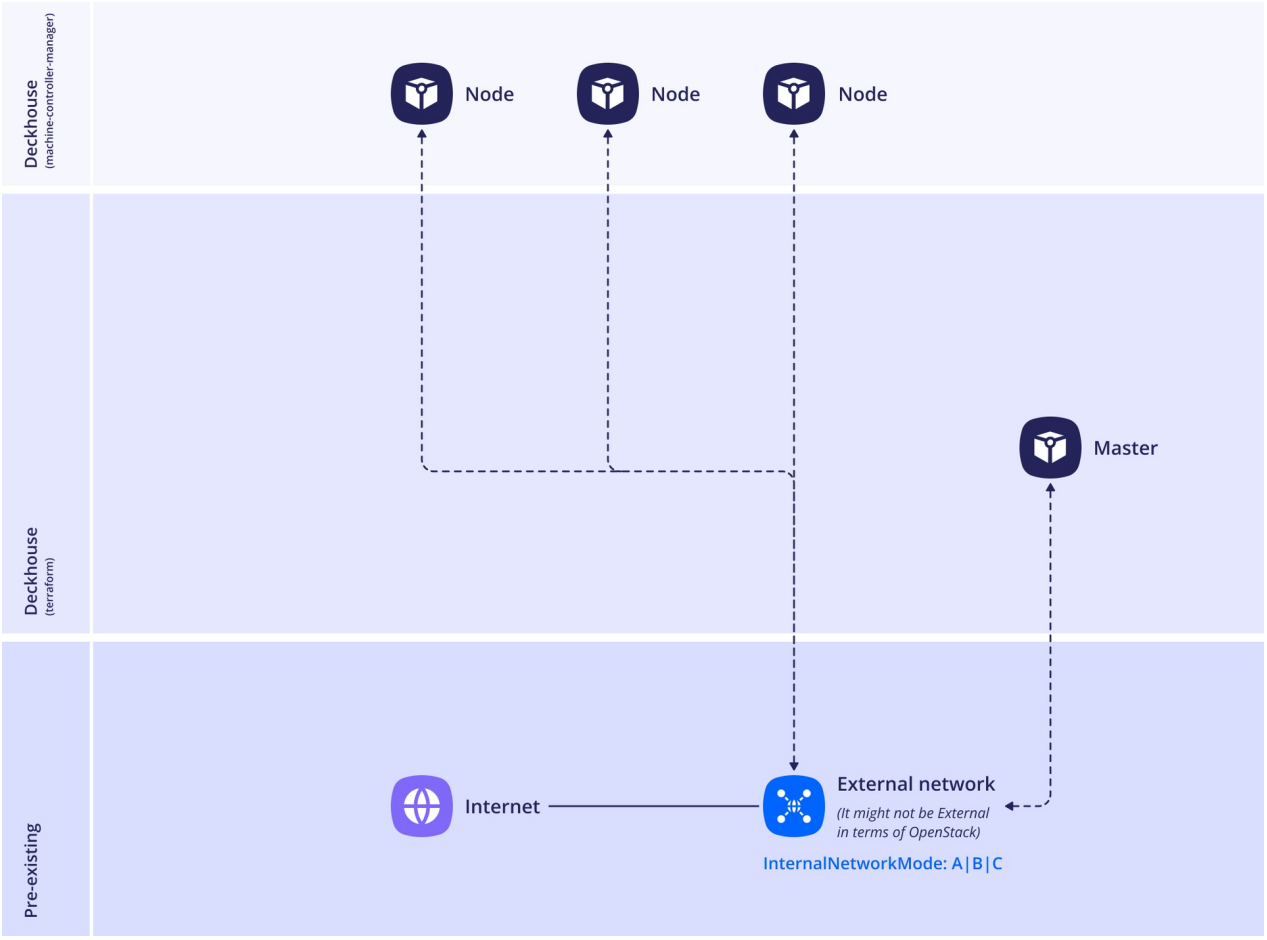
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, whether dhcp is enabled in specified external
  # network (default true).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, default true.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the prefix parameter in the global ModuleConfig.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy might come in handy if you need to merge a Kubernetes cluster with existing VMs.

> **Caution!** This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a load balancer with the Floating IP. An internal load balancer with the virtual IP in the public network is only accessible to cluster nodes.



In this layout, the module does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

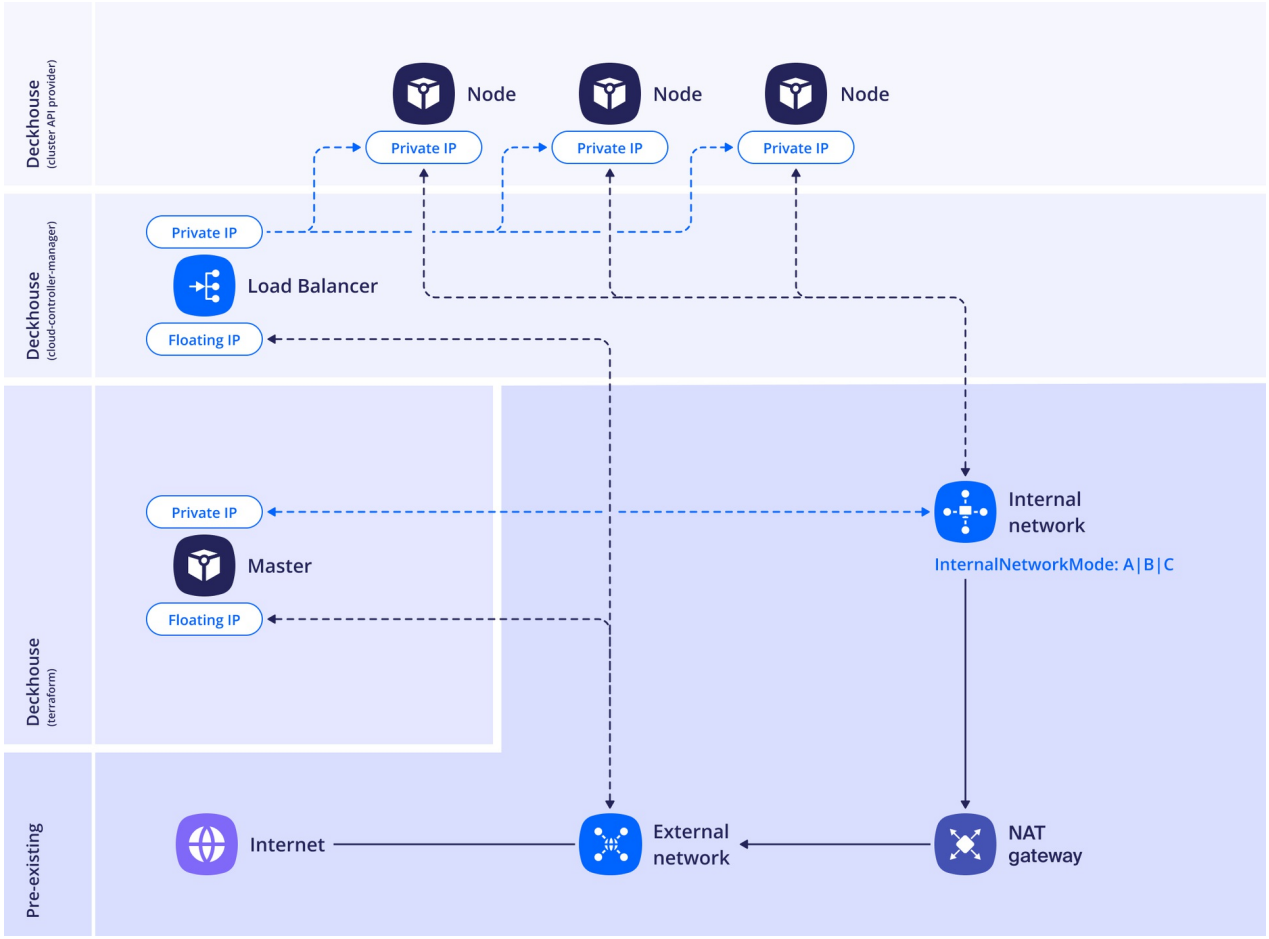
- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, default true.
  # Optional, by default VXLAN, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, network will be used as default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This placement strategy might come in handy if you need to merge a Kubernetes cluster with existing VMs.



In this layout, the module does not create security groups. Prepare them in the cloud beforehand by following the OpenStack documentation and attach them to the nodes via `additionalSecurityGroups`:

- For master nodes, in the `masterNodeGroup.instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For static nodes, in the `nodeGroups[].instanceClass.additionalSecurityGroups` parameter of the OpenStackClusterConfiguration resource.
- For ephemeral nodes, in the `spec.additionalSecurityGroups` parameter of the OpenStackInstanceClass resource.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # All cluster nodes have to be in the same subnet. Set either the subnet name
  # in internalSubnetName or its UUID in internalSubnetID.
  internalSubnetName: pivot-standard
  # Optional, by default DirectRoutingWithPortSecurityEnabled,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional, if set will be used for load balancer default
  # configuration and ordering master floating IP.
  externalNetworkName: ext-net
  # Optional, default value is true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, network will be used as default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — VMware Cloud Director

> Included in extensions: **On-prem IaaS**

> The module lifecycle stage: General Availability

The `cloud-provider-vcd` module integrates Deckhouse Platform with VMware Cloud Director. It allows the `node-manager` module to use VCD resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-vcd` module:

- Managing VCD resources via `cloud-controller-manager`:
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in VCD;
  - creates load balancers for Services of the LoadBalancer type. This uses VMware NSX Advanced Load Balancer (Avi); support is available with NSX-T.
- Provisioning disks via the Named Disk CSI driver (`named-disk.csi.cloud-director.vmware.com`) so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-vcd`. Base infrastructure is provisioned depending on the layout.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the VCDInstanceClass resource.
- Registering with `node-manager` so that VCDInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> For VCD API versions earlier than 37.2, the module uses compatibility mode (legacy components).

Third-party components

List of third-party software used in the `cloud-provider-vcd` module:

- VMware Cloud Director Cluster API provider v1.3.2, v1.2.0

  License: Apache License 2.0

  Cluster API provider for VMware Cloud Director.

- VMware Cloud Director Cloud provider 1.5.0, 1.6.1

  License: Apache License 2.0

  Cloud Controller Manager for VMware Cloud Director.

- VMware Cloud Director CSI driver 1.4.1, 1.6.0

  License: Apache License 2.0

  CSI driver for VMware Cloud Director Named Disks.

- VMware Cloud Director Terraform provider 3.10.0, 3.14.1

  License: Mozilla Public License 2.0

  Terraform provider for VMware Cloud Director.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Included in extensions: **On-prem IaaS**

> The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in VMware Cloud Director.

List of required VMware Cloud Director services

The following VMware Cloud Director service must be available for Deckhouse Platform to operate:

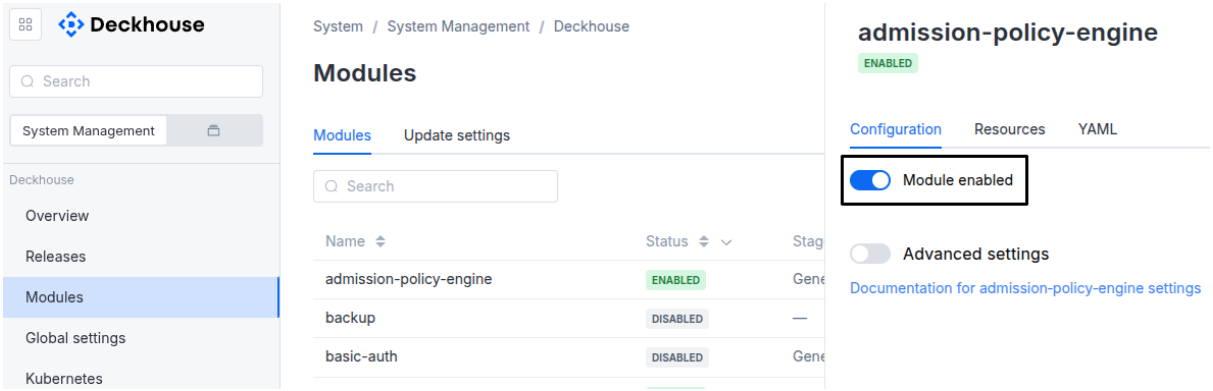
Service	API version
VMware Cloud Director API	`37.2` and later

> For VMware Cloud Director API versions earlier than `37.2`, compatibility mode for legacy API versions is used.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vcd` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-vcd
```

- **Using ModuleConfig `cloud-provider-vcd`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-vcd` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-vcd`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vcd
spec:
  enabled: true
```

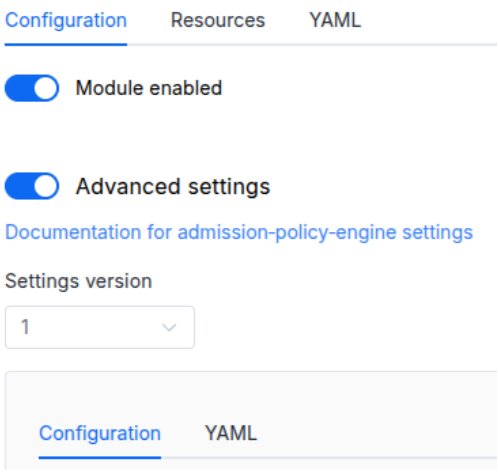
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vcd` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cloud-provider-vcd` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-vcd`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-vcd` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-vcd
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-vcd` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-vcd`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-vcd
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.mainNetwork**
    string

    Path to the network to which the primary network interfaces of virtual machines will be connected..

    Example:

    ```
    mainNetwork: internal
    ```
  - **settings.metadata**
    object

    Custom metadata entries that will be stored in cluster objects, such as virtual machines, disks, and networks.

    > **i** Changing metadata will cause CloudEphemeral nodes to be recreated.
  - **settings.organization**
    string

    Name of the Organization in VCD.

- **settings.provider**
  object

  Parameters for connecting to the VCD API.

  - **settings.provider.apiToken**
    string

    Token for authentication in VCD.

    > **i** When using the `apiToken` parameter, the `username` and `password` parameters must be empty.

  - **settings.provider.insecure**
    boolean

    Set to `true` if VCD has a self-signed certificate.

    Default: `false`

  - **settings.provider.password**
    string

    VCD user password.

  - **settings.provider.server**
    string

    VCD server host or IP address.

  - **settings.provider.username**
    string

    VCD user name with full permissions for the project.

- **settings.sshPublicKey**
  string

  Public SSH key for accessing the nodes.

- **settings.storageClass**
  object

  The module automatically creates StorageClasses that are available in VCD (VMware Cloud Director).

  - **settings.storageClass.default**
    Deprecated
    string

    The name of StorageClass that will be used in the cluster by default.

    If the parameter is omitted, the default StorageClass is either:

    - an arbitrary StorageClass present in the cluster that has the default annotation;
    - StorageClass for `default StorageProfile`;
    - the first (in lexicographic order) StorageClass created by the module.

    > **i** **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

    Example:

    ```
    default: ceph-ssd
    ```

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-hdd"
    - iscsi-fast
    ```

- **settings.virtualApplicationName**
  string

  Name of the Virtual Application in VCD that belongs to the specified Virtual Data Center.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.virtualDataCenter**
  string

  Name of the Virtual Data Center in VCD that belongs to the specified Organization.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

> If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vcd` module in the Deckhouse configuration. Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VCDClusterConfiguration structure for configuration.

VCDClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in VCD.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    template: Org/Templates/ubuntu-focal-20.04
    sizingPolicy: 16cpu32ram
    storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`
- **bastion**
  object

  Parameters for the bastion host, which is used to access the cluster's nodes.

- **bastion.instanceClass**
  object

  Required value

  Partial contents of the fields of the VCDInstanceClass.

  - **bastion.instanceClass.additionalMetadata**
    object

    Custom string key-value pairs to be stored in the Metadata of the bastion, useful for organizing and categorizing your virtualization inventory.

    The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

  - **bastion.instanceClass.mainNetworkIPAddress**
    string

    The IP address of the bastion host in the `mainNetwork`. If not set, the DHCP client will be enabled.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **bastion.instanceClass.placementPolicy**
    string

    Placement policy to be used on this machine.

    If no placement policy is specified, the default placement policy will be used to create the bastion host.

    Length: `1..128`

  - **bastion.instanceClass.rootDiskSizeGb**
    integer

    Root disk size in gigabytes.

    Default: `50`

    Allowed values: `0 <= X`

    Example:

    ```
    rootDiskSizeGb: 50
    ```

  - **bastion.instanceClass.sizingPolicy**
    string

    Required value

    Sizing policy to be used on this machine.

    If no sizing policy is specified, the default sizing policy will be used to create the bastion host.

    Length: `1..128`

  - **bastion.instanceClass.storageProfile**
    string

    Required value

    Storage profile to be used on this machine.

    Length: `1..128`

  - **bastion.instanceClass.template**

    Required value

    Path of the template OVA that is to be used.

    Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

    If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

    You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

    Length: `1..128`

- **createDefaultFirewallRules**
  boolean

  If enabled, configures the following default firewall rules for the cluster’s main network (mainNetwork):

  - Allows all outgoing connections from mainNetwork.
  - Allows all incoming ICMP packets to mainNetwork.
  - Allows all incoming TCP connections to mainNetwork on port 22 (SSH).

  - Allows all incoming TCP and UDP connections to mainNetwork for NodePorts in the range 30000–32767.

Attention! Using this option together with an Edge Gateway of type NSX-T implies that only one cluster should be deployed per Edge Gateway, because in this case all existing firewall rules will be overwritten.

Default: `false`

- **edgeGateway**
  object

  VMware Cloud Director Edge Gateway parameters (belongs to Virtual Data Center).

  - **edgeGateway.NSX-V**
    object

    Additional parameters required if `type` is NSX-V.

    - **edgeGateway.NSX-V.externalNetworkName**
      string

      External network name for build DNAT rule for master node. The external network is the one connected to the Edge Gateway in `Gateway Interface` and has an external IP address.

      Length: `1..128`

    - **edgeGateway.NSX-V.externalNetworkType**
      string

      External network type for build DNAT rule for master node (one of “org” or “ext”).

      Allowed values: `org`, `ext`

  - **edgeGateway.externalIP**
    string

    Required value

    The IP address accessible from external networks through which incoming SSH connections are forwarded to the master node using DNAT.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **edgeGateway.externalPort**
    integer

    The external port mapped by the DNAT rule to forward incoming SSH connections to the master node.

    Default: `22`

    Allowed values: `1 <= X <= 65535`

    Example:

    ```
    externalPort: 22
    ```

  - **edgeGateway.name**
    string

    Required value

    VMware Cloud Director Edge Gateway name.

    Length: `1..128`

  - **edgeGateway.type**
    string

    Required value

    Network virtualization platform type providing the VMware Cloud Director Edge Gateway (one of “NSX-V” or “NSX-T”). The type can be obtained from your administrator.

    Allowed values: `NSX-V`, `NSX-T`

- **internalNetworkCIDR**
  string

  Subnet for master nodes in the internal network.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkDHCPPoolStartAddress**
  integer

  Specifies the starting address number in the internal network from which the DHCP address pool will begin. Addresses below this number will not be assigned automatically by DHCP and are available for manual allocation. If not set, the default start address is 30.

  For example, in a network 192.168.100.128/27, a value of 10 means the DHCP pool will start with the IP address 192.168.100.138. The pool will end at the address immediately before the broadcast address (192.168.100.159), i.e., the last assignable host address.

Allowed values: `2 <= X`

- **internalNetworkDNSServers**
  array of strings

  A list of addresses of the recursive DNSs servers.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **kind**
  string

  Required value

  Allowed values: `VCDClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **legacyMode**
  boolean

  Support for outdated versions of the VCD API.

  If set to `true`, the API version below `37.2` will be used.

  The parameter has an effect only when installing a cluster. In the working cluster, the need to enable the support for outdated versions of the VCD API is determined automatically.

  Default: `false`

- **mainNetwork**
  string

  Required value

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  Example:

  ```
  mainNetwork: internal
  ```

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the VCDInstanceClass.

    - **masterNodeGroup.instanceClass.additionalMetadata**
      object

      Custom string key-value pairs to be stored in the Metadata of every master node VM, useful for organizing and categorizing your virtualization inventory.

      The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    - **masterNodeGroup.instanceClass.affinityRule**
      object

      Specifies the affinity rule applied to virtual machines of the master nodes within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

      - **masterNodeGroup.instanceClass.affinityRule.polarity**
        string

        Required value

The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

Allowed values: `Affinity`, `AntiAffinity`

- **masterNodeGroup.instanceClass.affinityRule.required**
  boolean

  Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

  Default: `false`

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in gigabytes.

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.mainNetworkIPAddresses**
  array of strings

  A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

  By default, the DHCP client is enabled.

  Example:

  ```
  mainNetworkIPAddresses:
  - 10.1.14.20
  - 10.1.4.21
  ```

  - **Element of the array**
    string

    An IP address with a CIDR prefix.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **masterNodeGroup.instanceClass.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine.

  If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in gigabytes.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **masterNodeGroup.instanceClass.sizingPolicy**
  string

  Required value

  SizingPolicy is the sizing policy to be used on this machine.

  If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

  ```
  sizingPolicy: 4Cpu8Memory
  ```

- **masterNodeGroup.instanceClass.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

```
storageProfile: ssd
```

- **masterNodeGroup.instanceClass.template**
  string

  Required value

  Path of the template OVA that is to be used.

  Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

  If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

  You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

  Examples:

  ```
  template: Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: MyOrg/Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

  ```
  template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **metadata**
  object

  Custom metadata entries to be stored in cluster objects such as virtual machines, disks and network.

  Clearing the `metadata` and `additionalMetadata` fields, if they contain any keys, will result in all metadata no longer being tracked except for nodes of the CloudEphemeral type. This means that if you need to remove metadata entirely, you will have to do it manually.

  > **Warning.** Changing the metadata will cause the recreation of CloudEphemeral nodes.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the VCDInstanceClass.

    - **nodeGroups.instanceClass.additionalMetadata**
      object

      Custom string key-value pairs to be stored in the Metadata of every node VM, useful for organizing and categorizing your virtualization inventory.

      The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    - **nodeGroups.instanceClass.affinityRule**
      object

      Specifies the affinity rule applied to virtual machines of nodes of the NodeGroup within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

      - **nodeGroups.instanceClass.affinityRule.polarity**
        string

        Required value

        The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that

nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

Allowed values: `Affinity`, `AntiAffinity`

- **nodeGroups.instanceClass.affinityRule.required**
  boolean

  Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

  Default: `false`

- **nodeGroups.instanceClass.mainNetworkIPAddresses**
  array of strings

  A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

  By default, the DHCP client is enabled.

  Example:

  ```
  mainNetworkIPAddresses:
  - 10.1.14.20
  - 10.1.4.21
  ```

  - **Element of the array**
    string

    An IP address with a CIDR prefix.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **nodeGroups.instanceClass.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine.

  If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **nodeGroups.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in gigabytes.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **nodeGroups.instanceClass.sizingPolicy**
  string

  Required value

  SizingPolicy is the sizing policy to be used on this machine.

  If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

  ```
  sizingPolicy: 4Cpu8Memory
  ```

- **nodeGroups.instanceClass.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

  ```
  storageProfile: ssd
  ```

- **nodeGroups.instanceClass.template**
  string

  Required value

  Path of the template OVA that is to be used.

  Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Examples:

```
template: Linux/ubuntu2204-cloud-ova
```

```
template: MyOrg/Linux/ubuntu2204-cloud-ova
```

```
template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

```
template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **organization**
  string

  Required value

  VMware Cloud Director Organization name.

- **provider**
  object

  Required value

  Parameters for connecting to the VCD.

  - **provider.apiToken**
    string

    The token for authentication.

    > **Caution!** When using `apiToken`, leave `username` and `password` empty.

  - **provider.insecure**
    boolean

    Set to `true` if VCD has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    The user's password.

  - **provider.server**
    string

    Required value

    The host or the IP address of the VCD server.

  - **provider.username**
    string

    The login ID.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **virtualApplicationName**
  string

  Required value

  VMware Cloud Director Virtual Application name (belongs to Virtual Data Center).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **virtualDataCenter**
  string

  Required value

  VMware Cloud Director Virtual Data Center name (belongs to Organization).

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> The module lifecycle stage: General Availability

List of required VCD resources

- Organization
- VirtualDataCenter
- vApp
- StoragePolicy
- SizingPolicy

- Network
- EdgeRouter
- Catalog

The Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter, and Catalog resources must be provided by your VMware Cloud Director service provider.

> ⚠ Each VDC (Virtual Data Center) must have an Edge Gateway configured, and the cluster network must be connected to it.

Network (internal network) can be configured either by your VMware Cloud Director service provider or manually by you. If you choose the `WithNAT` placement scheme, the network will be created automatically. The following section describes how to configure the internal network manually.

User permissions

For access to the VMware Cloud Director API, one of the following options is sufficient:

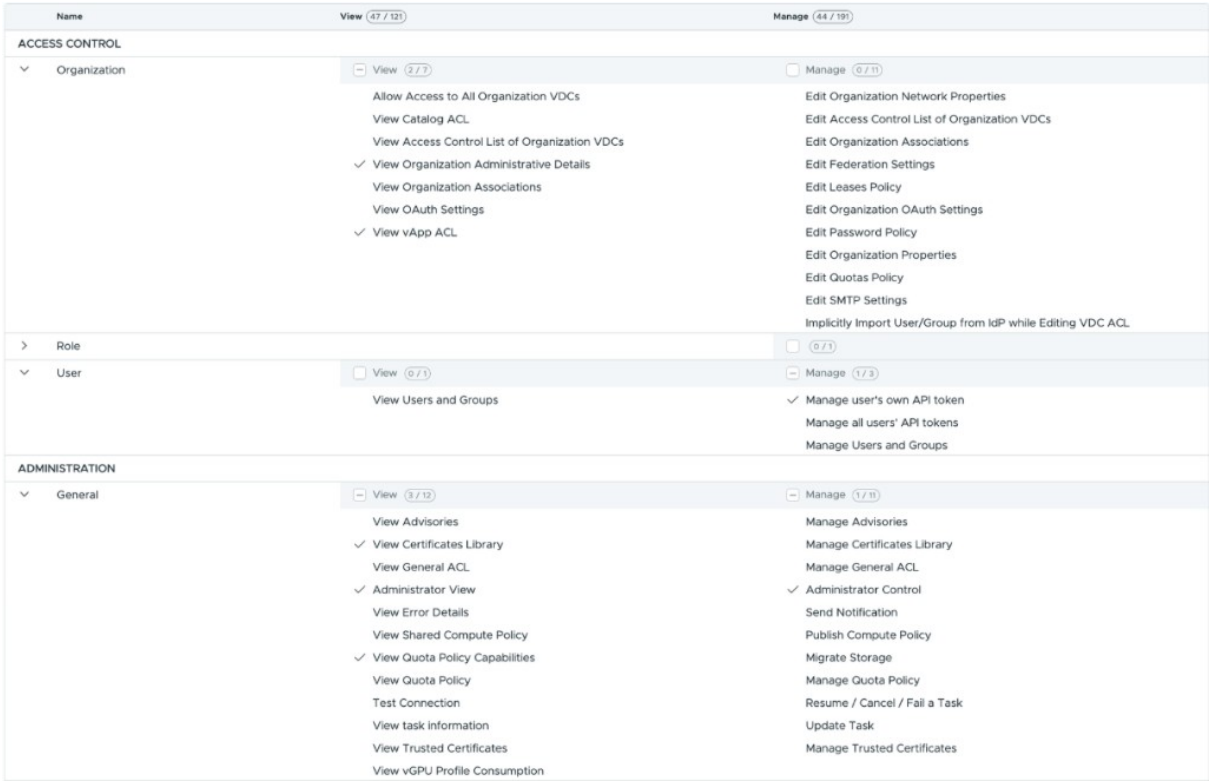
- The `Organization Administrator` role with the `Preserve All ExtraConfig Elements During OVF Import and Export` right. This option does not require configuring a separate role, but grants the user full administrative access to the organization.
- A custom role with the set of rights listed below. This set is sufficient to create a cluster, provision nodes (including CloudEphemeral with static addressing and DHCP), and work with CSI disks and StorageClass without granting additional administrative rights.

In VMware Cloud Director, user permissions come from the role and from a **Rights Bundle** — a set of rights published to the organization. Add the `Preserve All ExtraConfig Elements During OVF Import and Export` right both to the role and to the organization's Rights Bundle.

> ⚠ Without the `Preserve All ExtraConfig Elements During OVF Import and Export` right in both the role and the organization's Rights Bundle, `userdata` is not passed to the virtual machines of ephemeral nodes.

To create a custom role, enable the rights as shown below. Right names match the VCD UI.

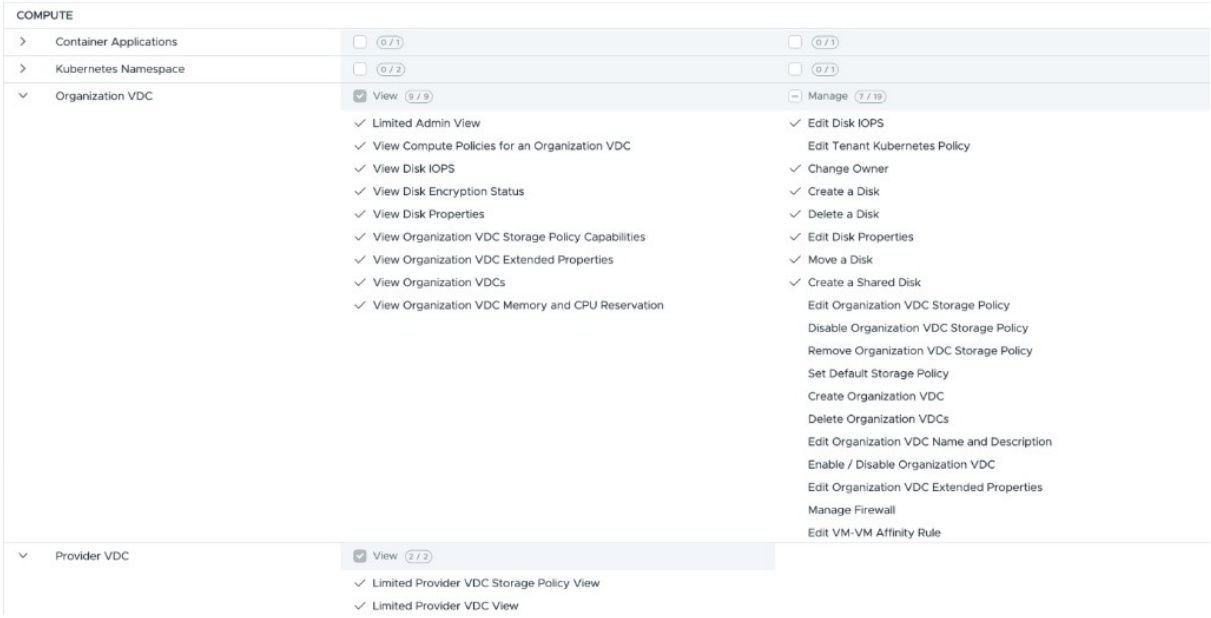
1. In the "ACCESS CONTROL" and "ADMINISTRATION" sections, enable the following rights:
   - Organization → View: `View Organization Administrative Details`, `View vApp ACL`
   - User → Manage: `Manage user's own API token`
   - General → View: `View Certificates Library`, `Administrator View`, `View Quota Policy Capabilities`
   - General → Manage: `Administrator Control`



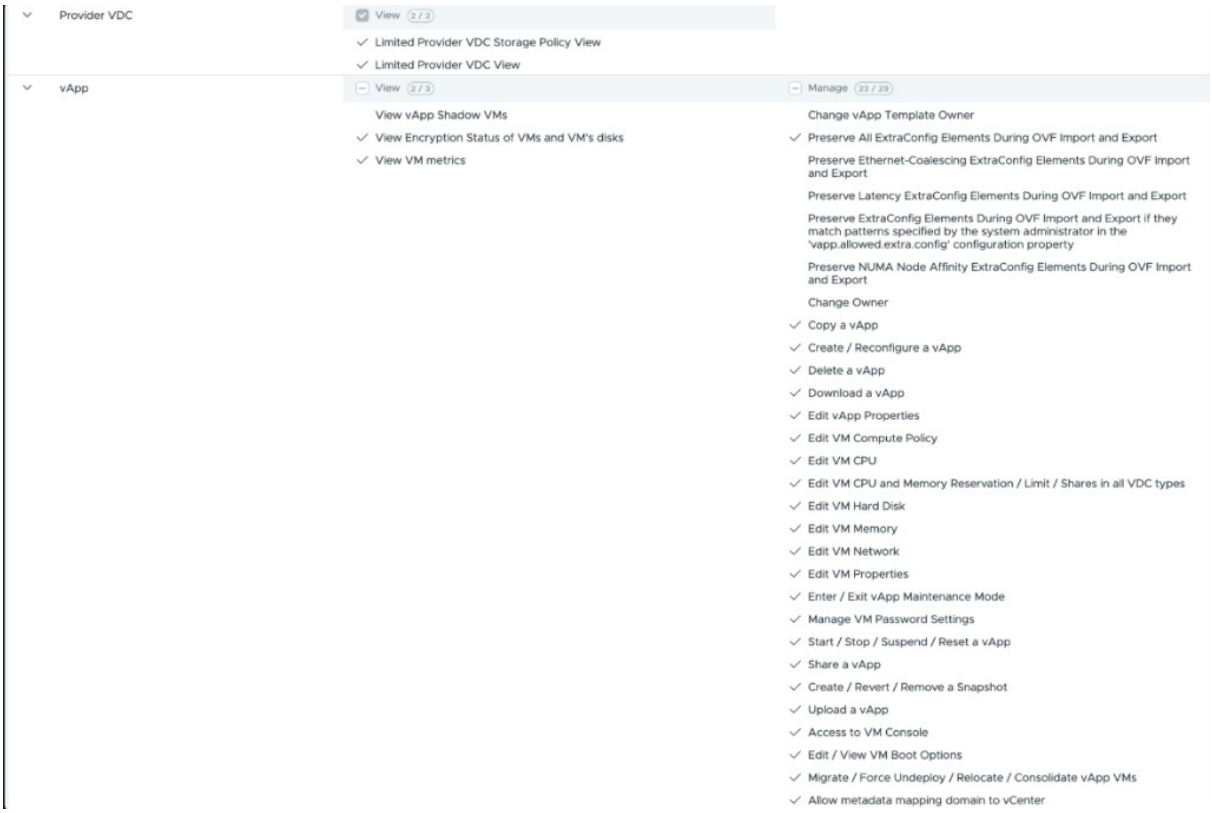
2. In the "COMPUTE" → "Organization VDC" section, enable all View rights and the following Manage rights:
   - `Edit Disk IOPS`
   - `Edit Tenant Kubernetes Policy`
   - `Change Owner`
   - `Create a Disk`
   - `Delete a Disk`
   - `Edit Disk Properties`

- `Move a Disk`
- `Create a Shared Disk`

In the “Provider VDC” → View section, enable `Limited Provider VDC Storage Policy View` and `Limited Provider VDC View`:



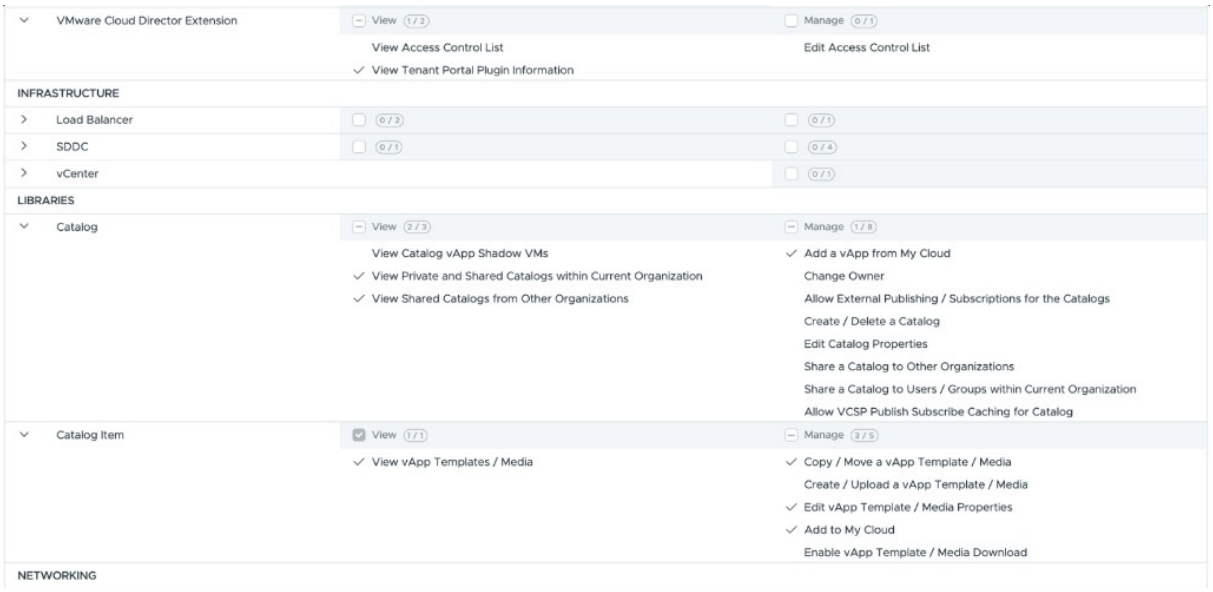
3. In the “COMPUTE” → “vApp” section, enable the rights as shown in the screenshot. Make sure that `Preserve All ExtraConfig Elements During OVF Import and Export` is enabled. In the verified role, View includes `View Encryption Status of VMs and VM's disks` and `View VM metrics`; `View vApp Shadow VMs` and the other Preserve ExtraConfig options are not enabled:



4. In the “LIBRARIES” section, enable the following rights:

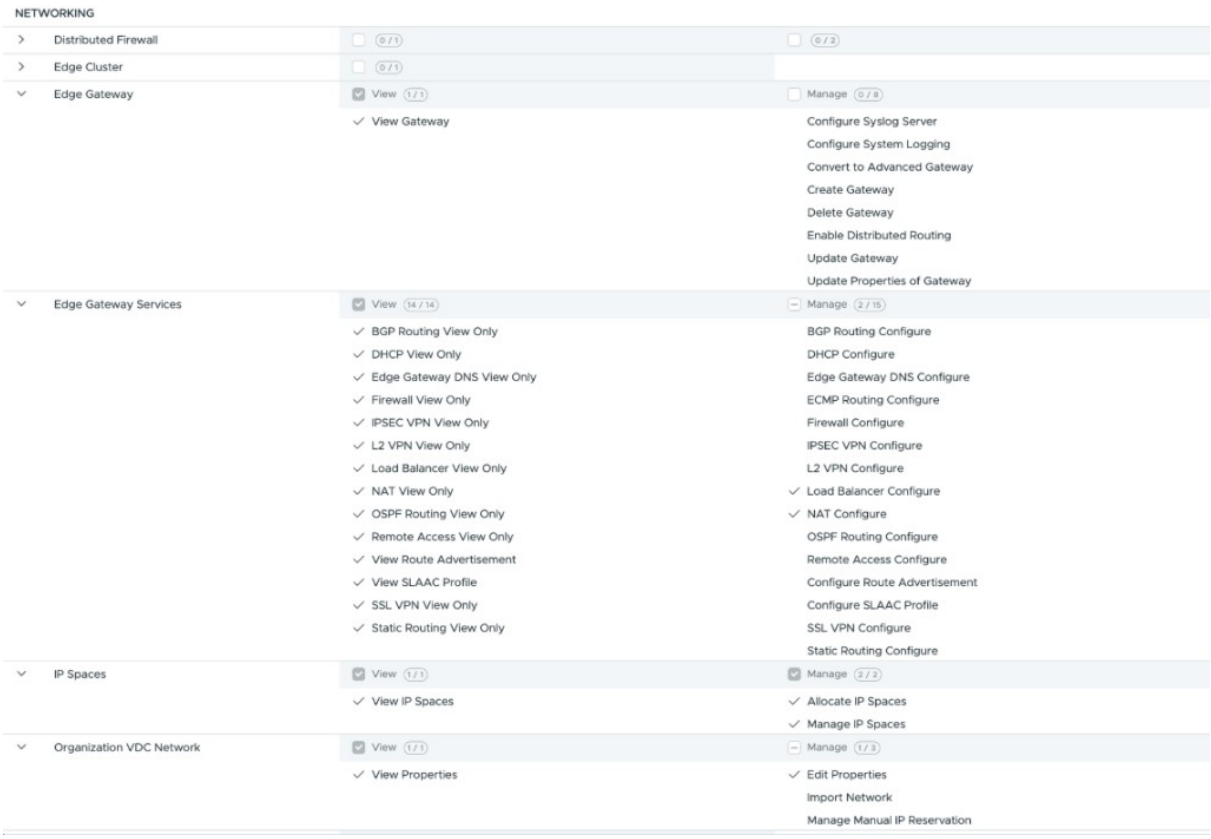
   - Catalog → View: `View Private and Shared Catalogs within Current Organization`, `View Shared Catalogs from Other Organizations` (to use images from shared catalogs)
   - Catalog → Manage: `Add a vApp from My Cloud`
   - Catalog Item → View: `View vApp Templates / Media`
   - Catalog Item → Manage: `Copy / Move a vApp Template / Media`, `Edit vApp Template / Media Properties`, `Add to My Cloud`

   In the “VMware Cloud Director Extension” → View section, enable `View Tenant Portal Plugin Information`:

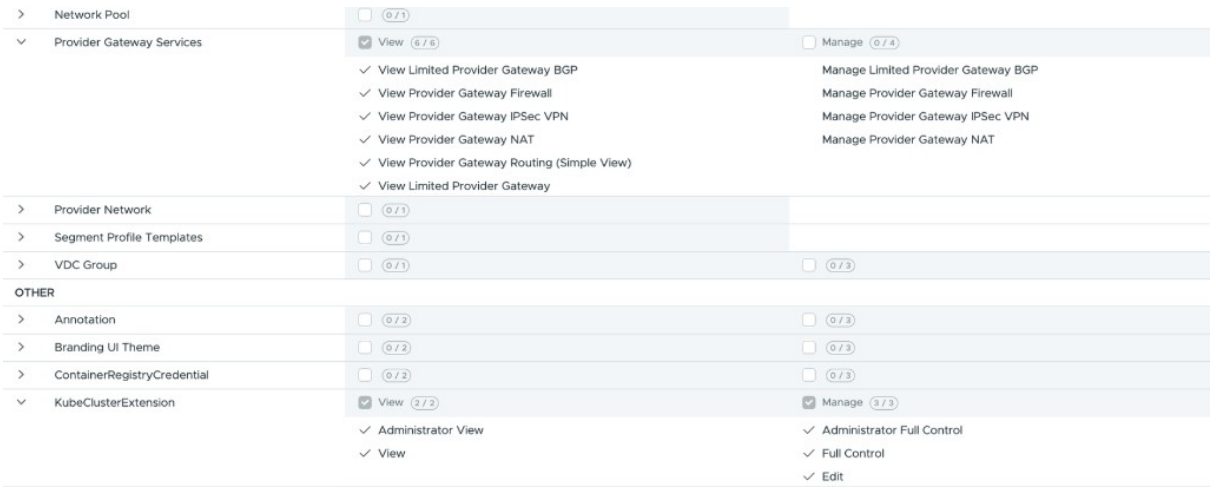


5. In the “NETWORKING” section, enable the following rights:

   - Edge Gateway → View: `View Gateway`
   - Edge Gateway Services → View: all items
   - Edge Gateway Services → Manage: `Load Balancer Configure`, `NAT Configure`
   - IP Spaces → View: `View IP Spaces`
   - IP Spaces → Manage: `Allocate IP Spaces`, `Manage IP Spaces`
   - Organization VDC Network → View: `View Properties`
   - Organization VDC Network → Manage: `Edit Properties` (required to create networks in the organization)



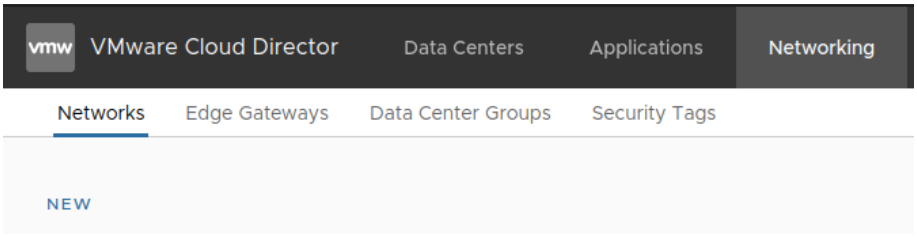
6. In the “Provider Gateway Services” section, enable all View rights. In the “KubeClusterExtension” section, enable all View and Manage rights:



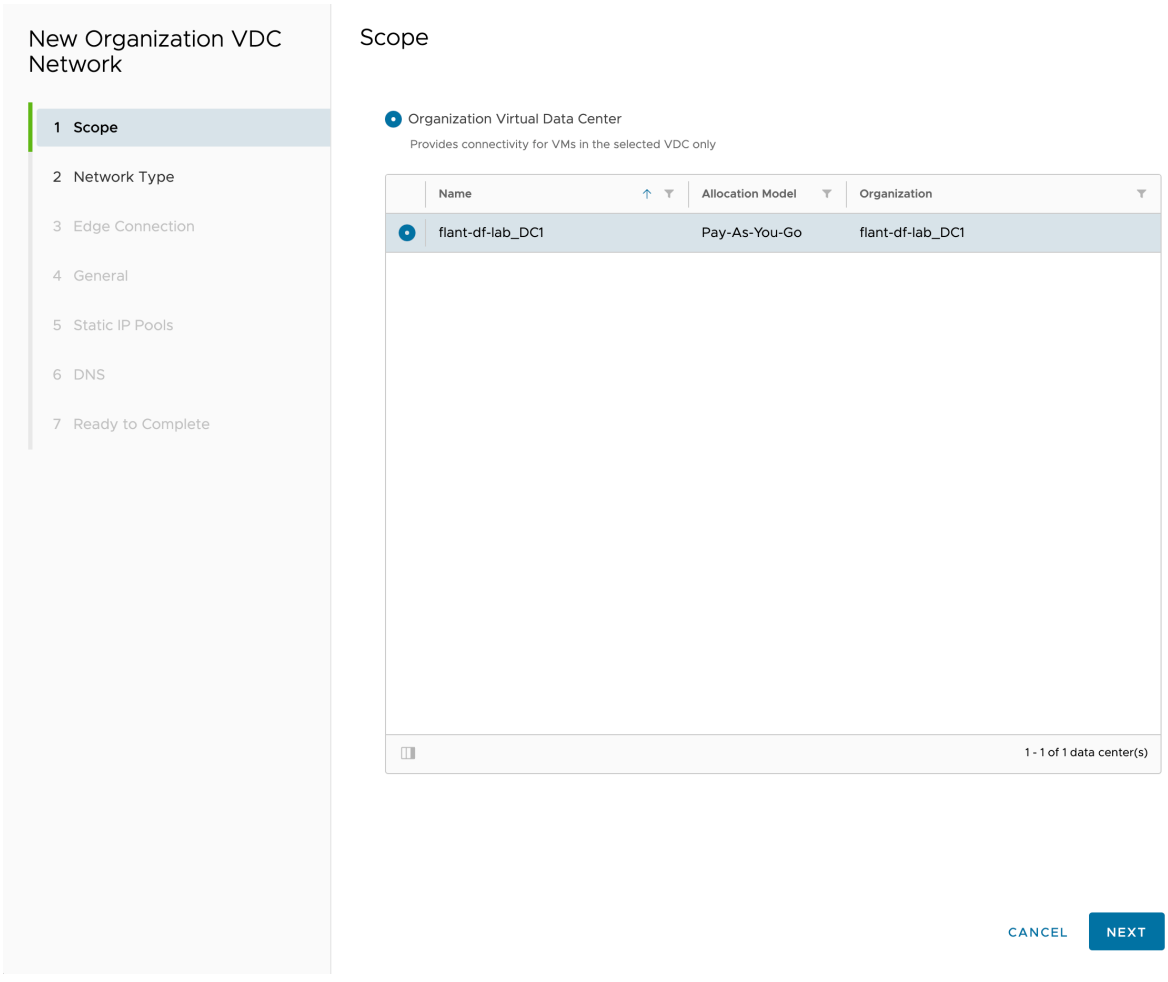
Adding a network

> **i** This instruction is applicable only for the `Standard` layout.

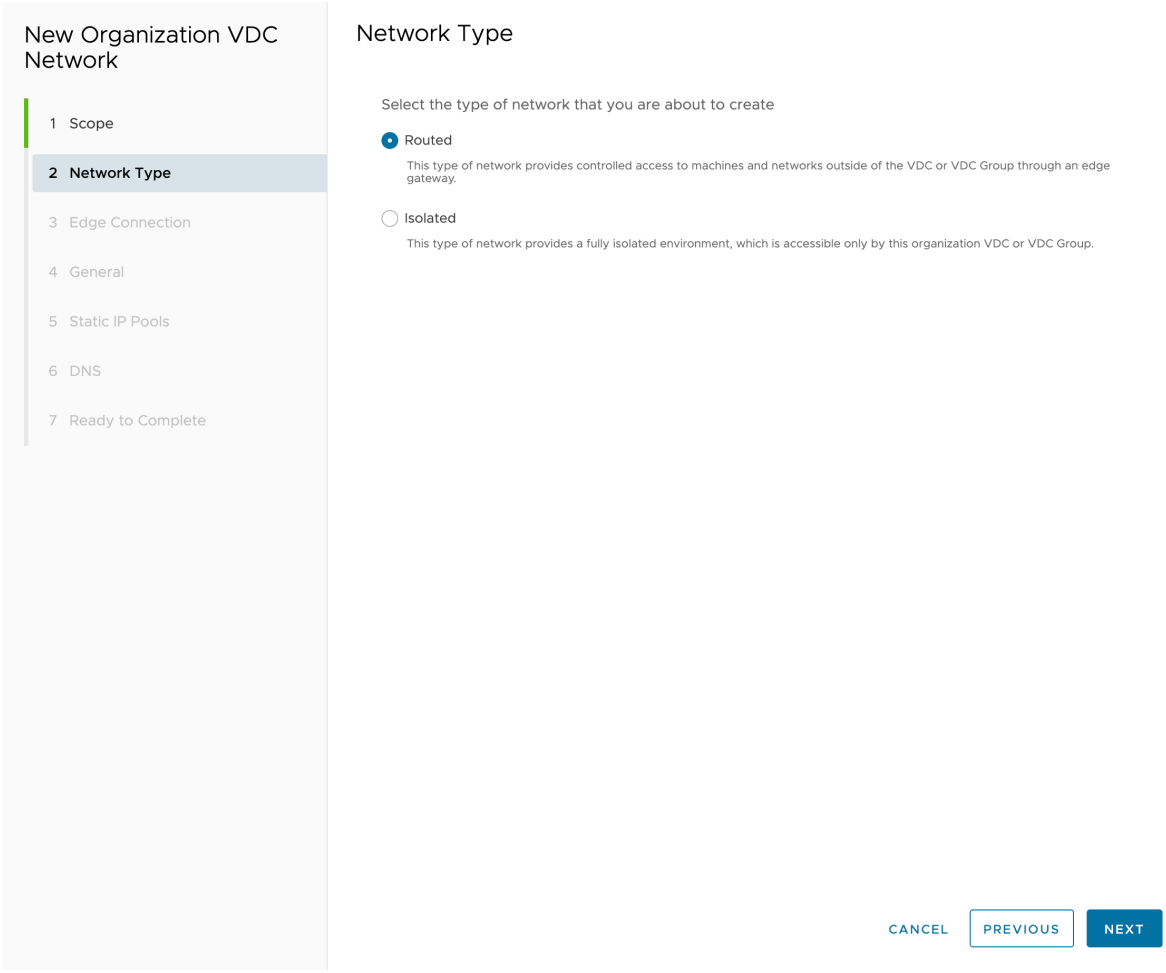
1. Go to the “Networking” tab and click “NEW”:



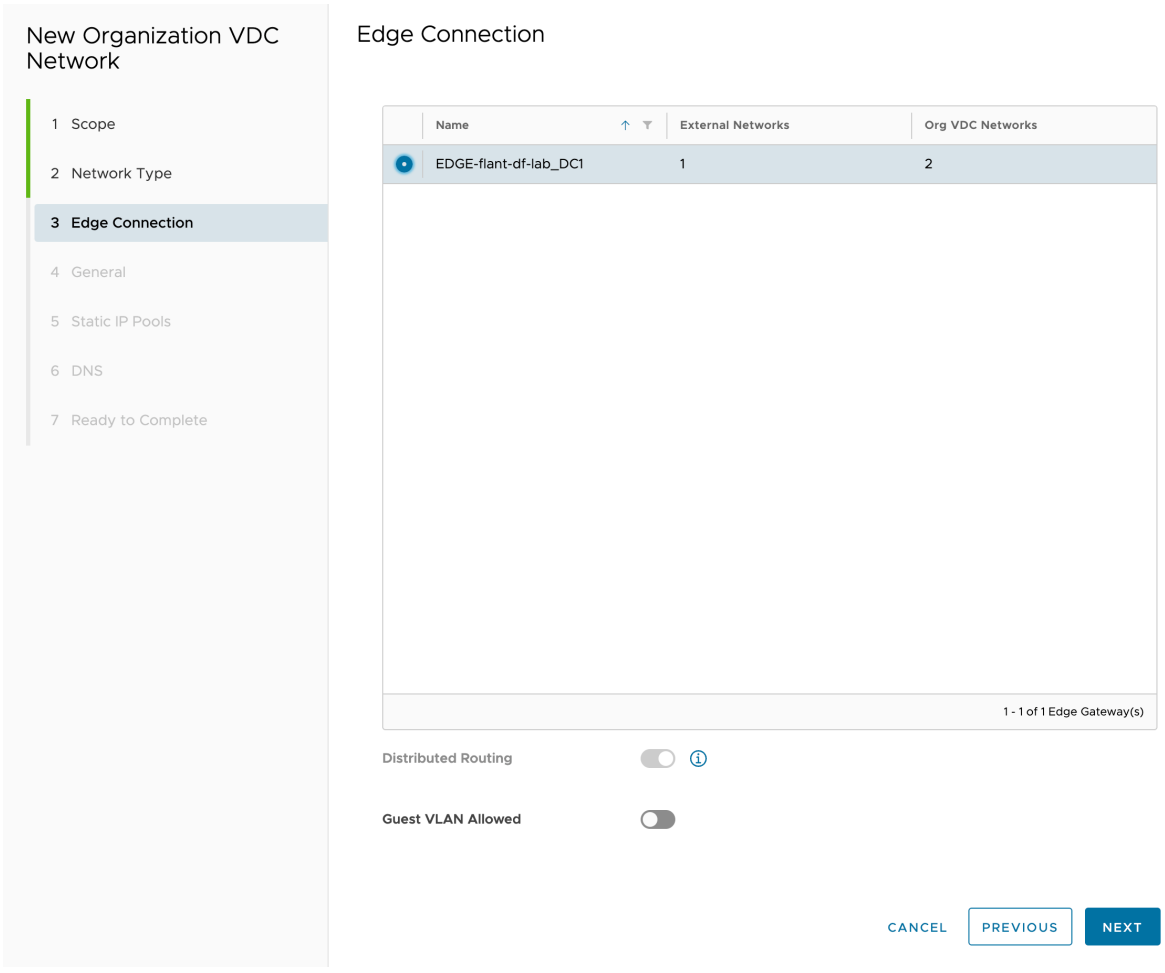
2. Select the Data Center:



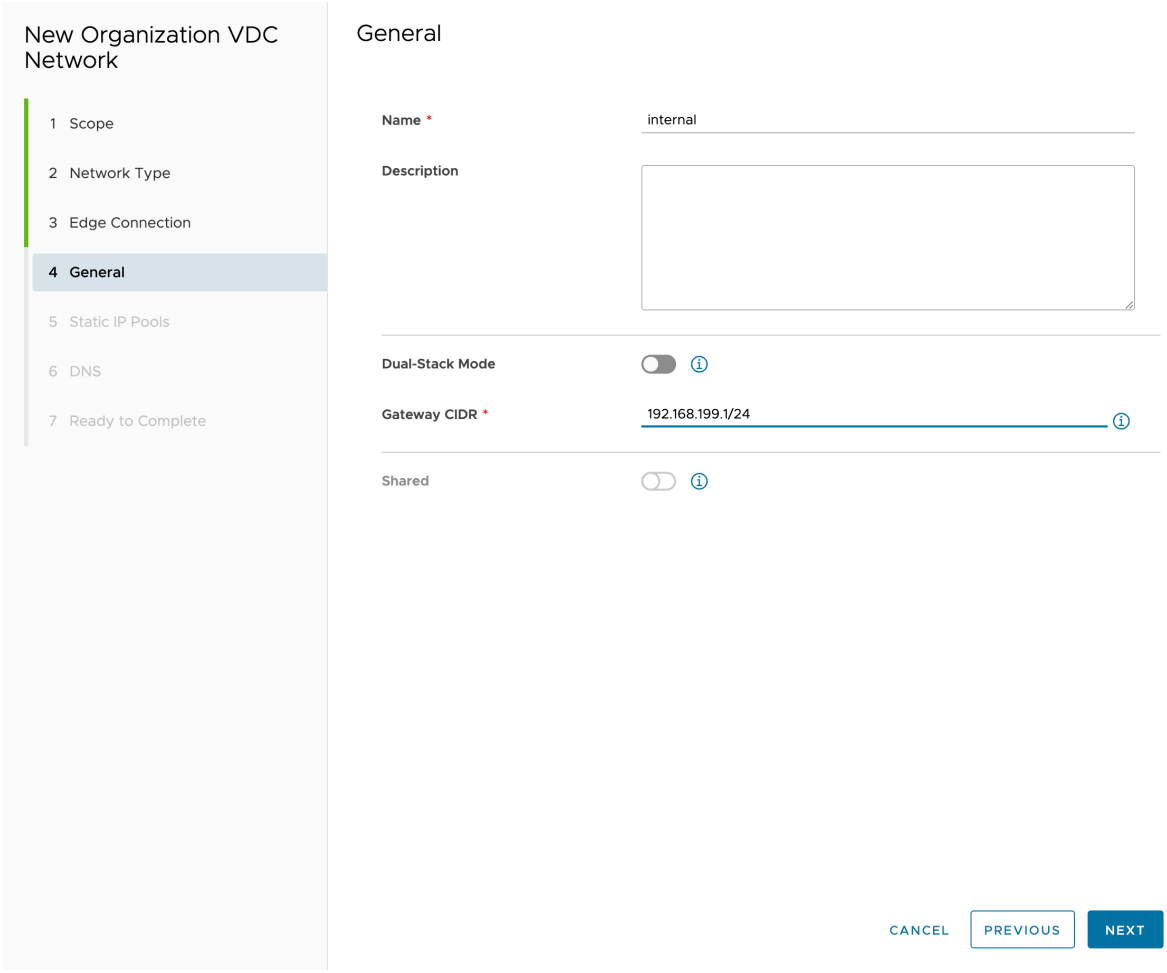
3. At the “Network type” step, select “Routed”:



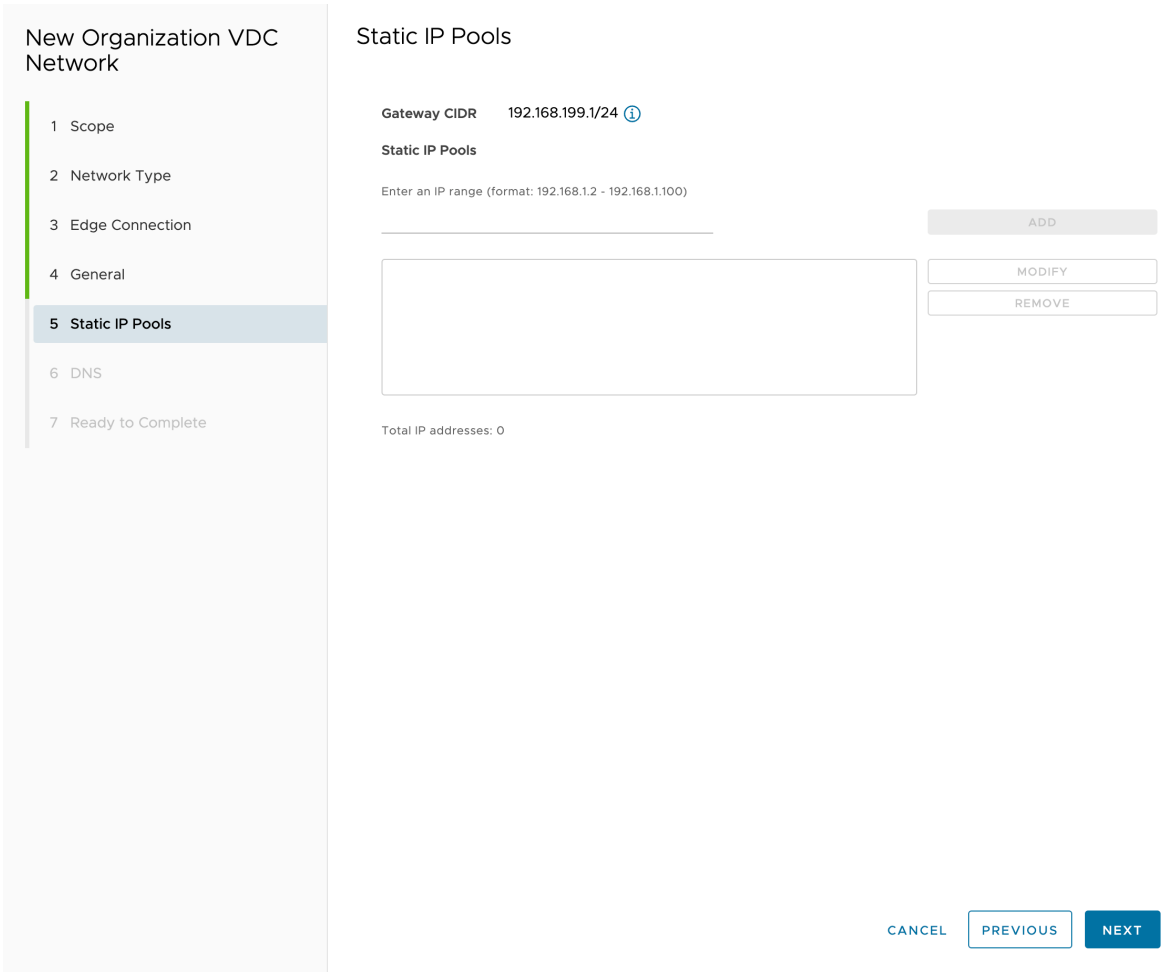
4. Connect `EdgeRouter` to the network:



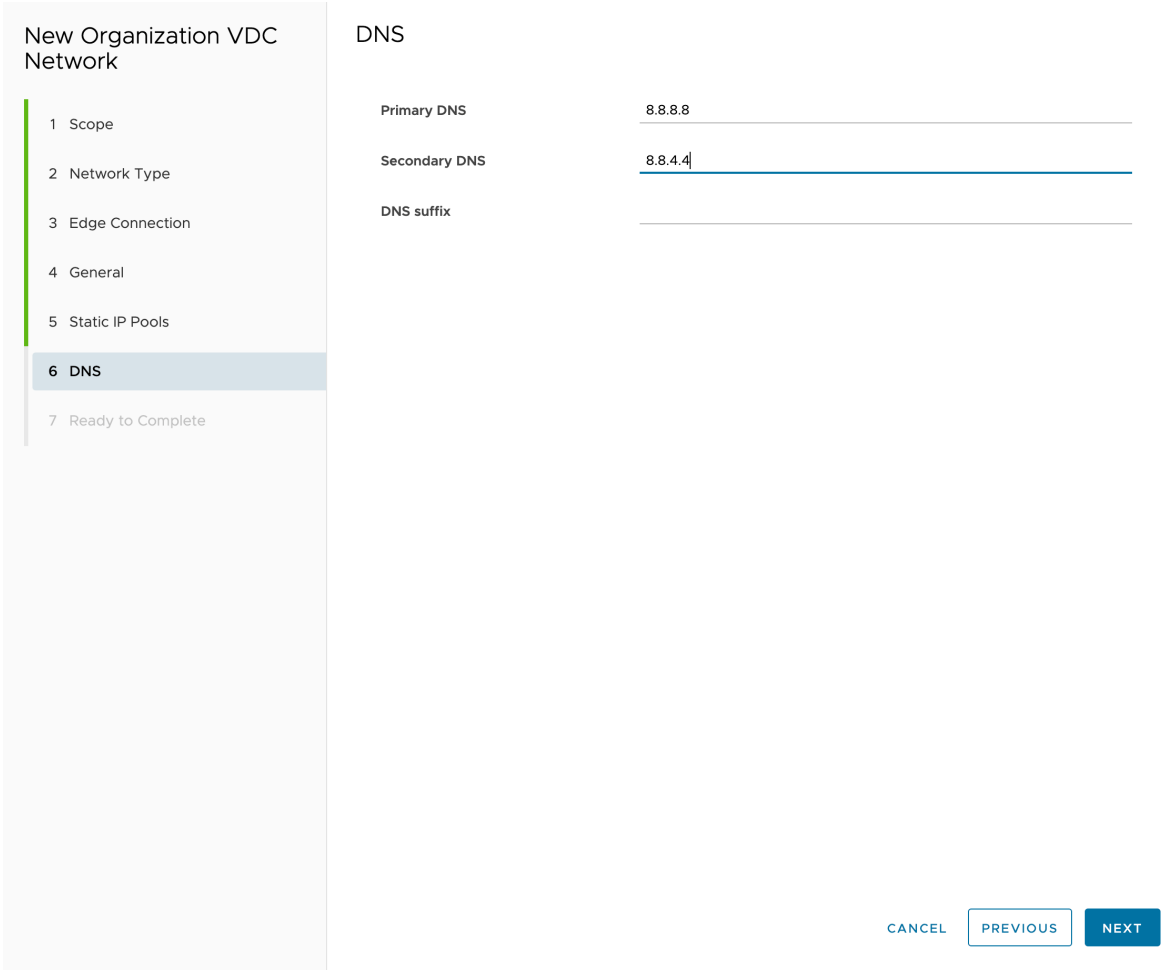
5. Specify the network name and CIDR:



6. Do not add “Static IP Pools” because DHCP will be used:



7. Specify the DNS server addresses:



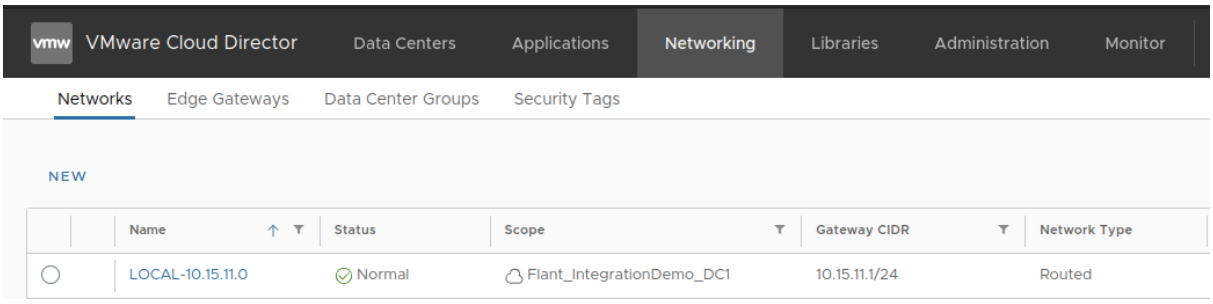
Configuring DHCP

> **i** This instruction is applicable only for the `Standard` layout.

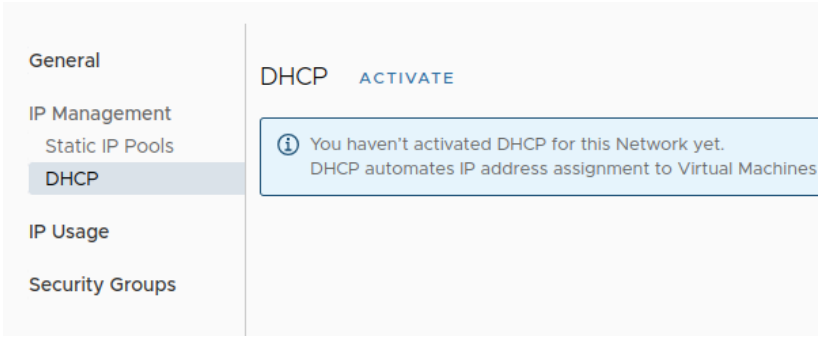
To provision nodes dynamically, enable the DHCP server for the internal network.

> **i** We recommend allocating the beginning of the network address range to system consumers (control plane, frontend nodes, system nodes) and the rest to the DHCP pool. For example, for a `/24` mask network it would be enough to allocate 20 addresses to system consumers.

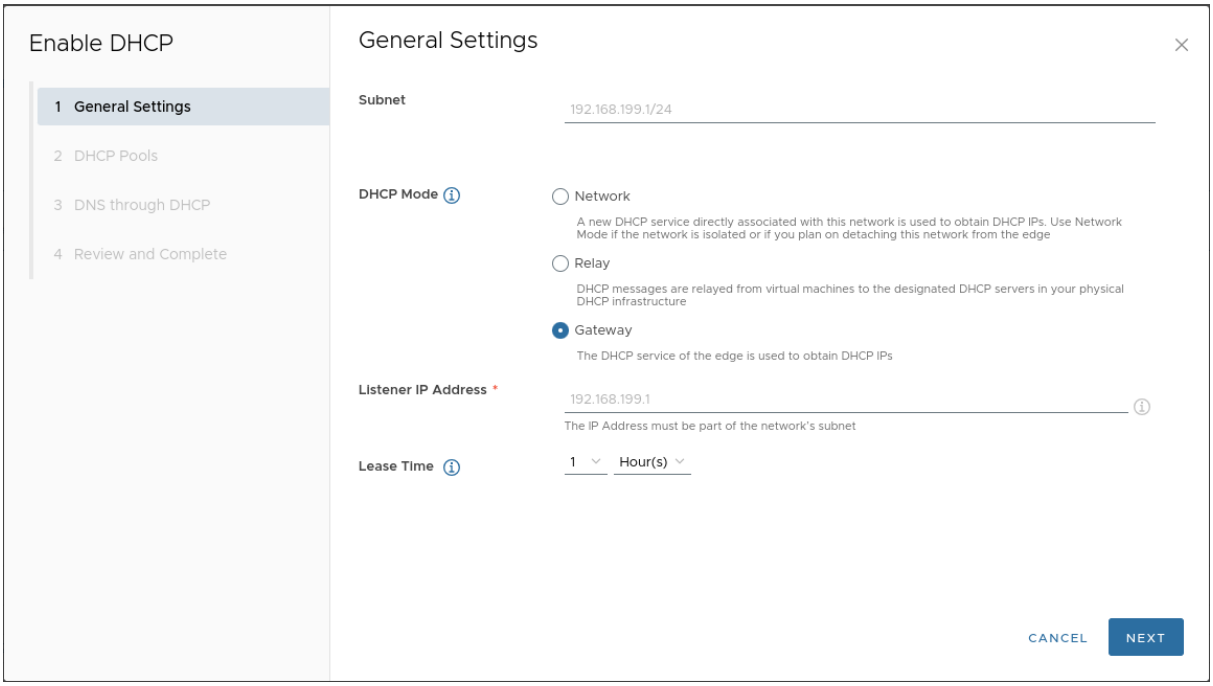
1. Click the “Networking” tab and open the network you created:



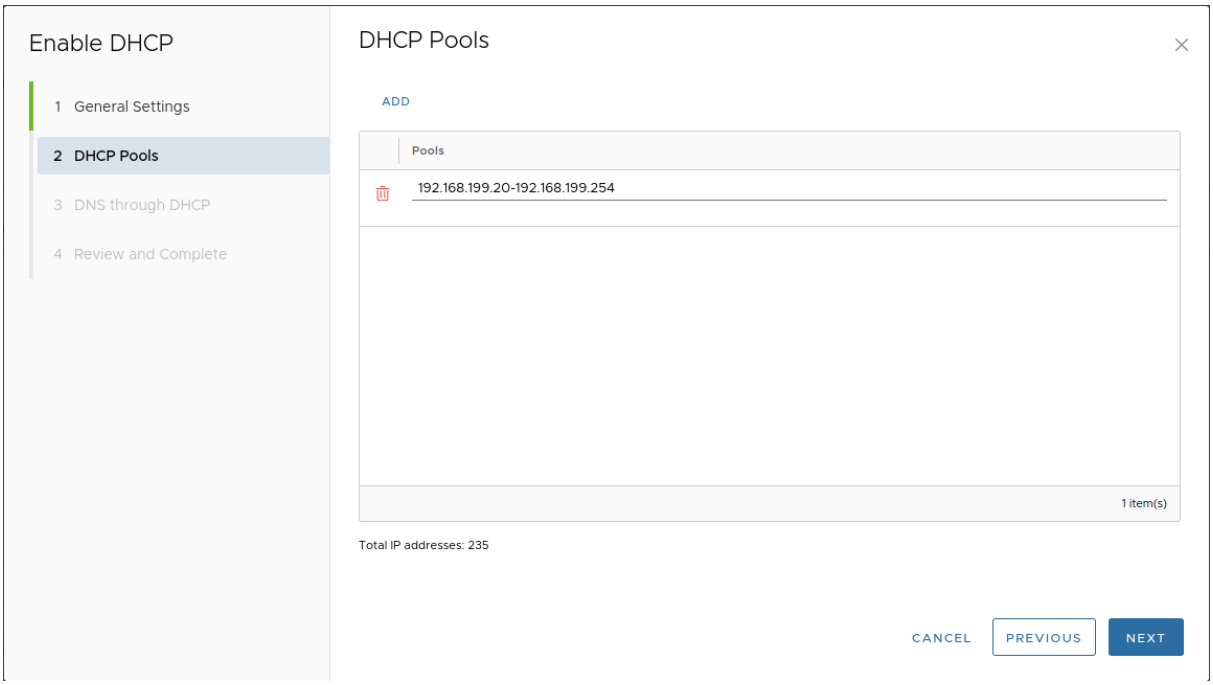
2. In the opened window, select “IP Management” → “DHCP” → “Activate”:



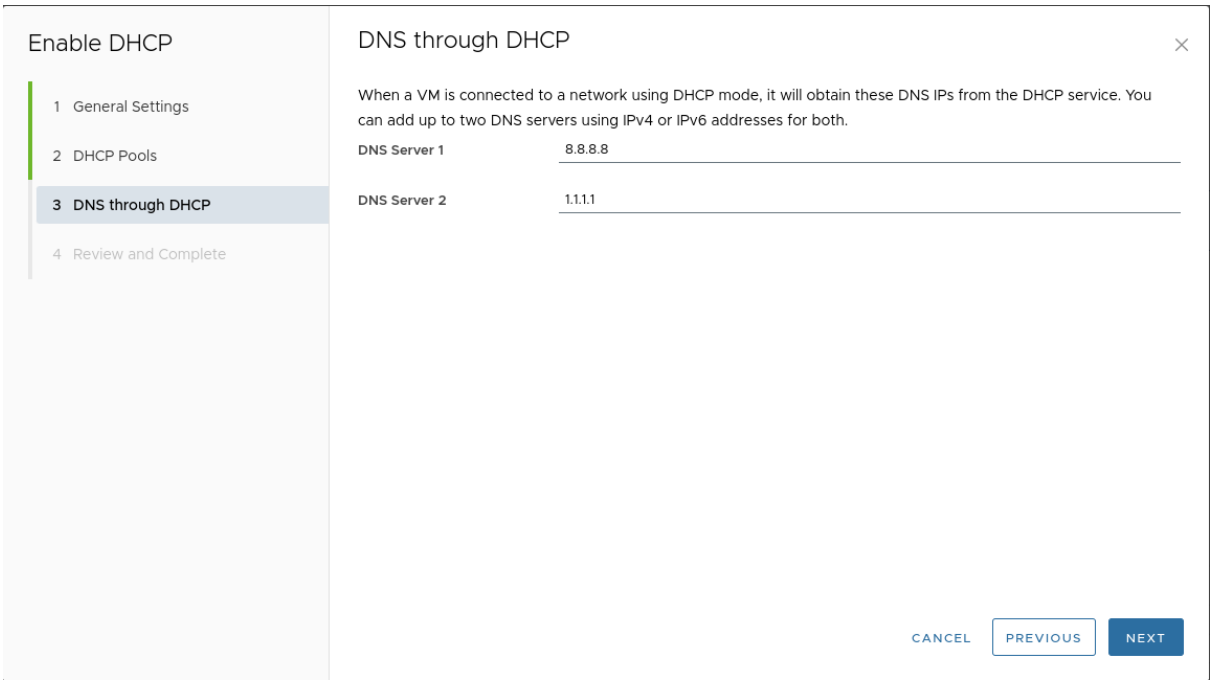
3. In the “General settings” tab, set the parameters as shown in the example:



4. Add a pool:



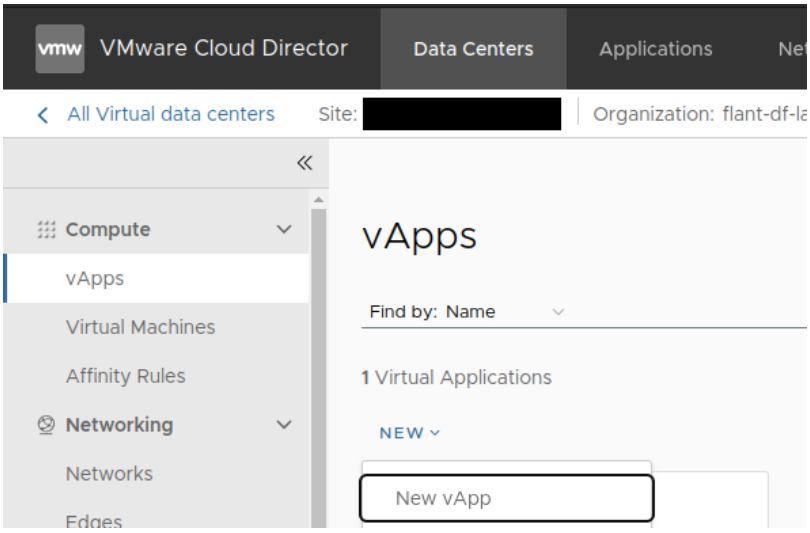
5. Set the DNS server addresses:



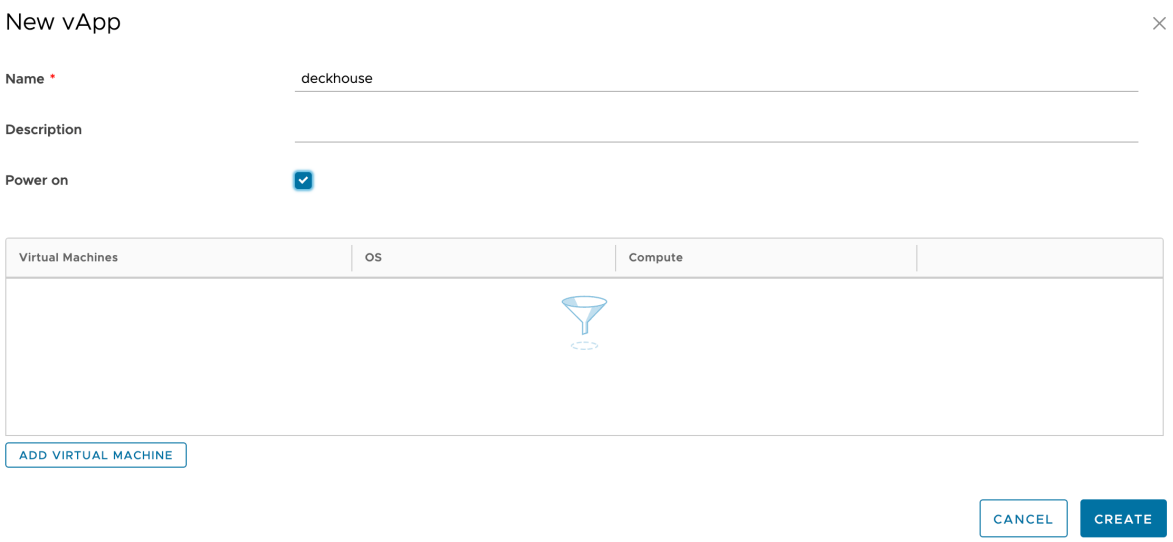
Adding a vApp

> **i** This instruction is applicable only for the `Standard` deployment layout.

1. Switch to the “Data Centers” tab → “vApps” → “NEW” → “New vApp”:



2. Specify a name and enable the vApp:

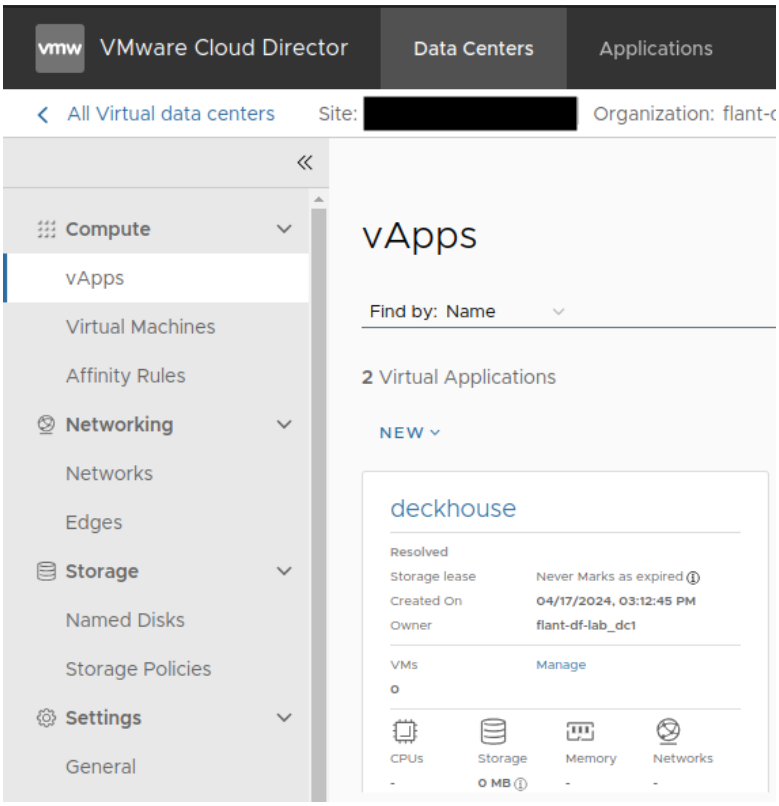


Adding a network to the vApp

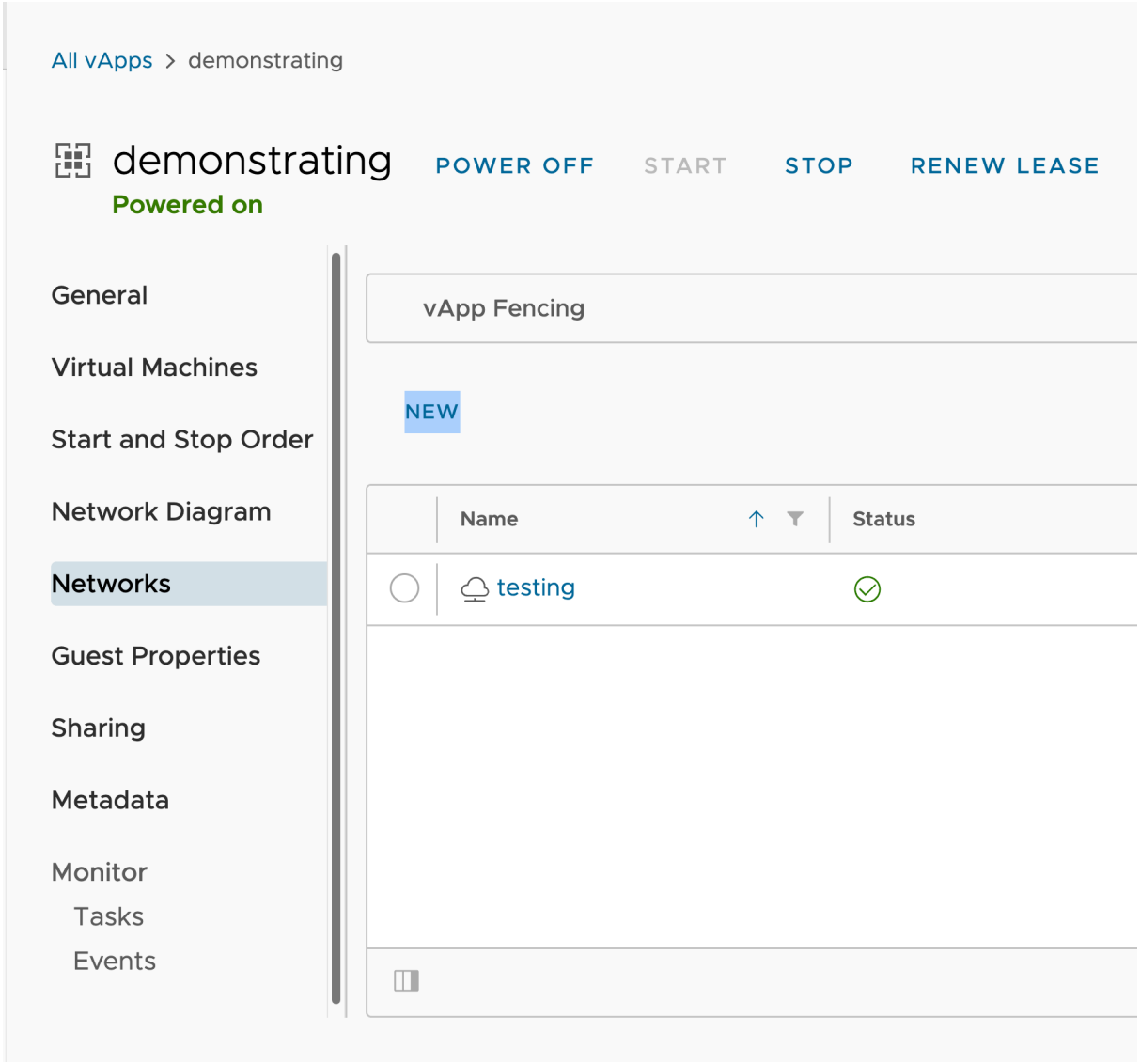
> **i** This instruction is applicable only for the `Standard` deployment layout.

Once the vApp is created, connect the created internal network to it.

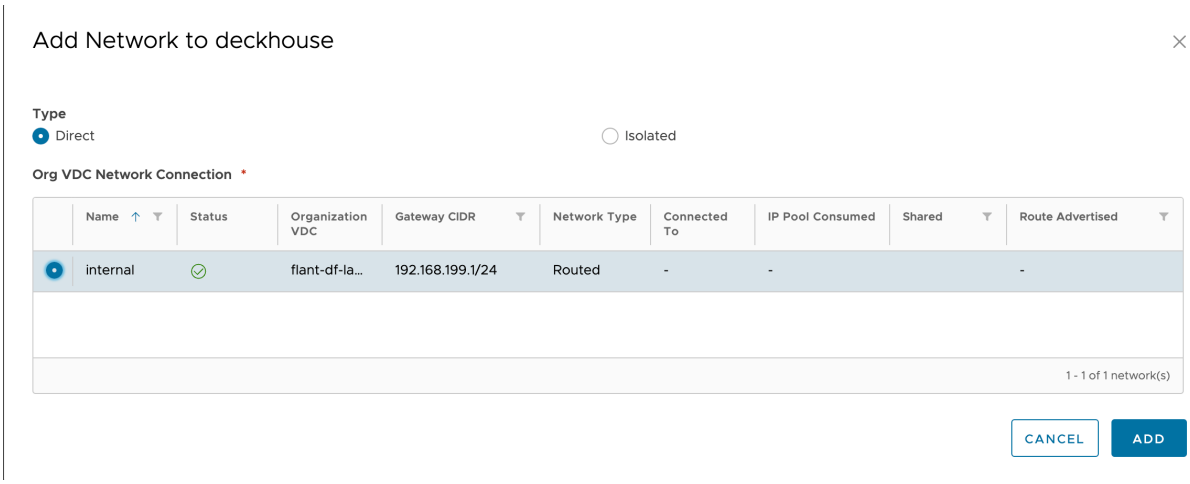
1. Switch to the “Data Centers” tab → “vApps” and open the target vApp:



2. Go to the “Networks” tab and click “NEW”:



3. In the opened window, click the “Direct” type and select the network:



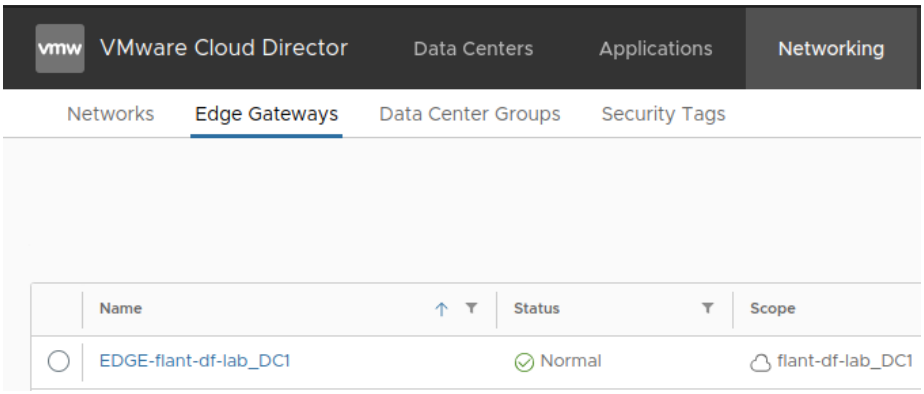
Incoming traffic

Incoming traffic should be routed to the edge router (ports `80`, `443`) using DNAT rules to be forwarded to a dedicated address on the internal network.
This address can be created by running MetalLB in L2 mode for dedicated frontend nodes.

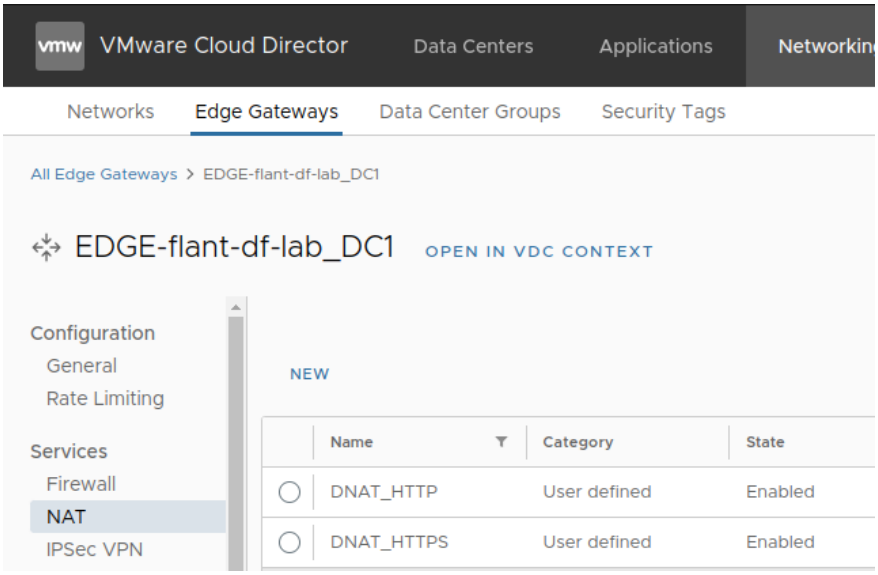
Configuring DNAT/SNAT rules on the edge gateway

> **i** This instruction is applicable only for the `Standard` deployment layout.

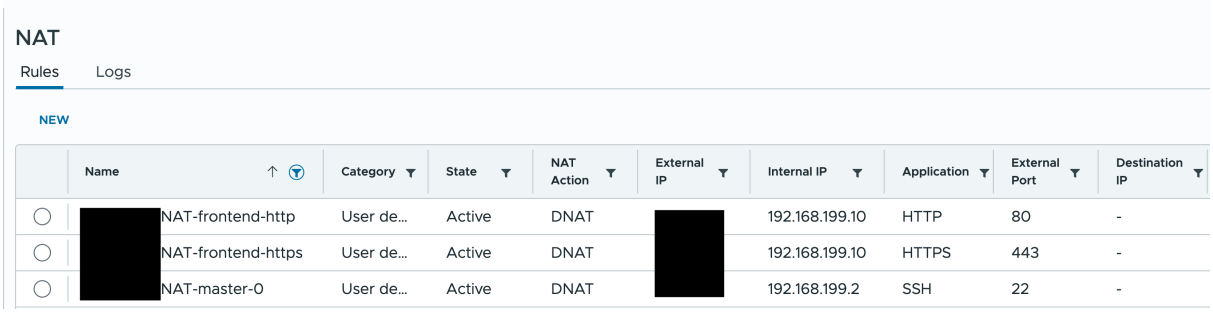
1. Navigate to the “Networking” tab → “Edge Gateways” and open the edge gateway:



2. Switch to the “Services” tab → “NAT”:

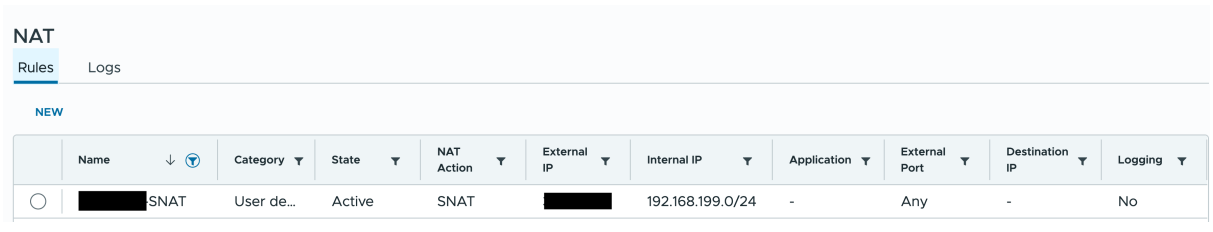


3. Add the following rules:



   The first two rules are used for incoming traffic, while the third rule is used for SSH access to the control plane host (without this rule the installation will not be possible).

4. To allow virtual machines to access the Internet, configure SNAT rules following the example:



   This rule will allow virtual machines from the `192.168.199.0/24` subnet to access the Internet.

Let’s Encrypt certificate issuance

> **i** This instruction is applicable only for the `Standard` deployment layout.

When using `cert-manager` with the HTTP-01 ACME challenge type, you may need to configure Hairpin NAT (NAT loopback) on the Edge Gateway.

This is required when the Ingress domain name resolves to the external IP address of the Edge Gateway, while requests to

that address are made from **inside** the cluster (from the node network). In this case, the traffic must be correctly routed back to the internal network to the Ingress address (for example, the MetalLB address).

This is important for issuing Let's Encrypt certificates because cert-manager performs a preliminary self-check of the challenge URL before sending a request to the ACME provider. If the URL is not reachable from inside the cluster, certificate issuance will not start, even if the address is accessible from outside.

Example (cluster network `192.168.199.0/24`):

- internal node network: `192.168.199.0/24`
- external Edge Gateway IP address: `194.117.83.19`
- internal Ingress address (for example, MetalLB): `192.168.199.251`.

In this case, you must configure Hairpin NAT for traffic from the `192.168.199.0/24` network that is sent to the external IP `194.117.83.19`, with translation to the internal Ingress address `192.168.199.251`.

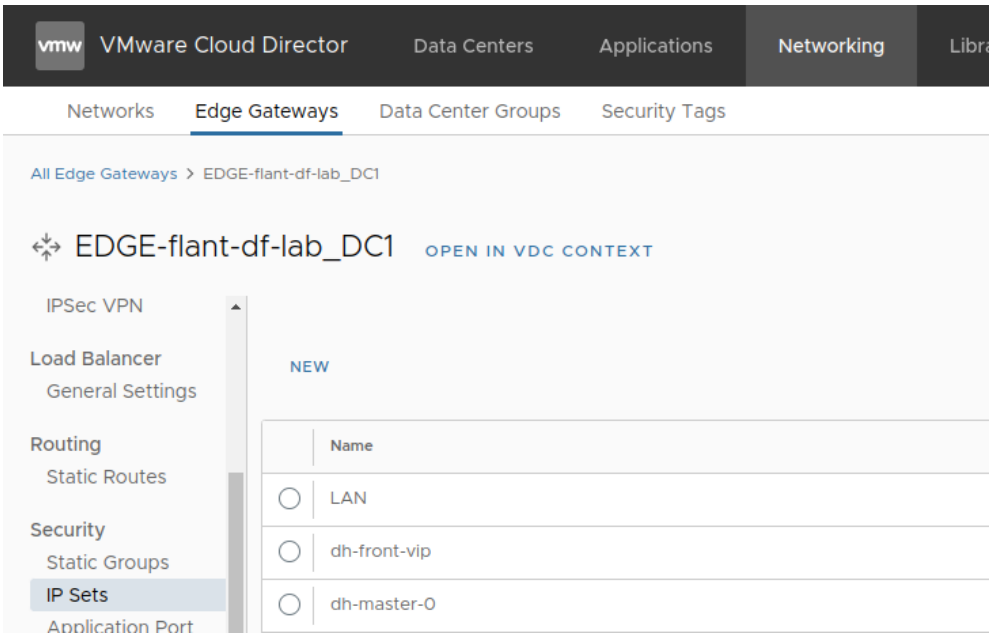
> **i** Hairpin NAT configuration may be required for Edge Gateway based on NSX-V.
>
> For NSX-T, the loopback scenario is often supported by default, but the actual behavior depends on the VMware Cloud Director provider configuration.

Configuring a firewall

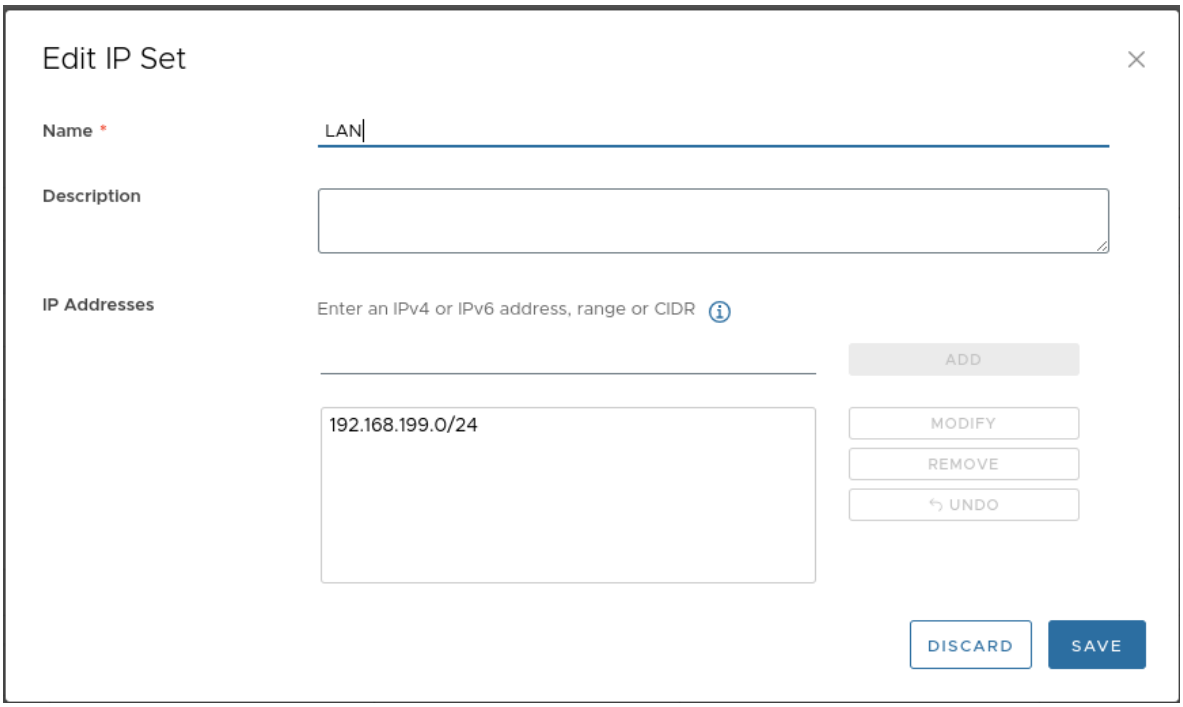
> **i** This instruction is applicable only for the `Standard` deployment layout.

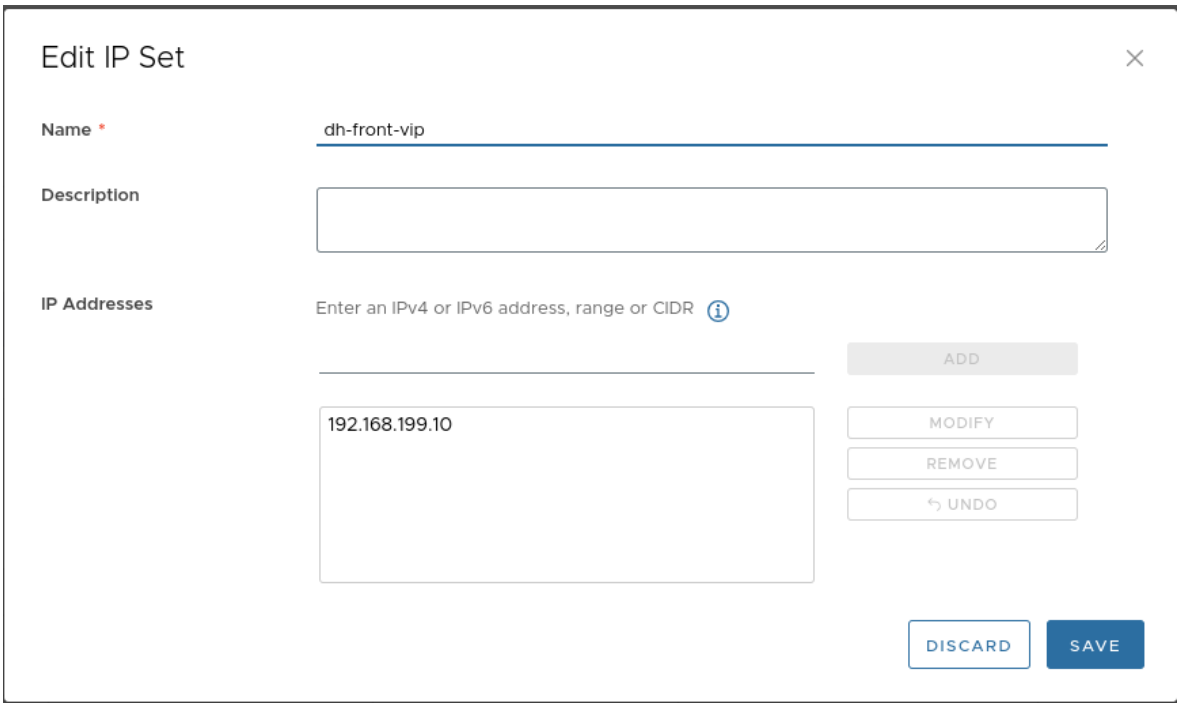
Once DNAT is configured, set up the firewall. Start by configuring the IP sets.

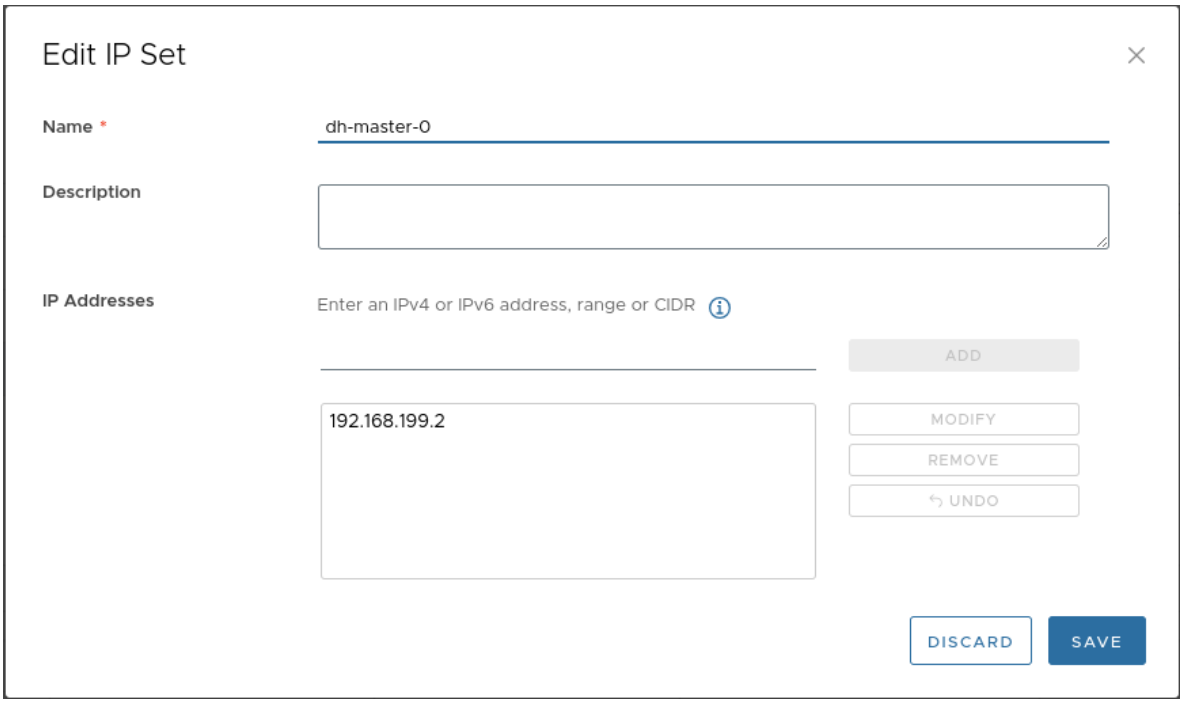
1. Switch to the "Security" tab → "IP Sets":



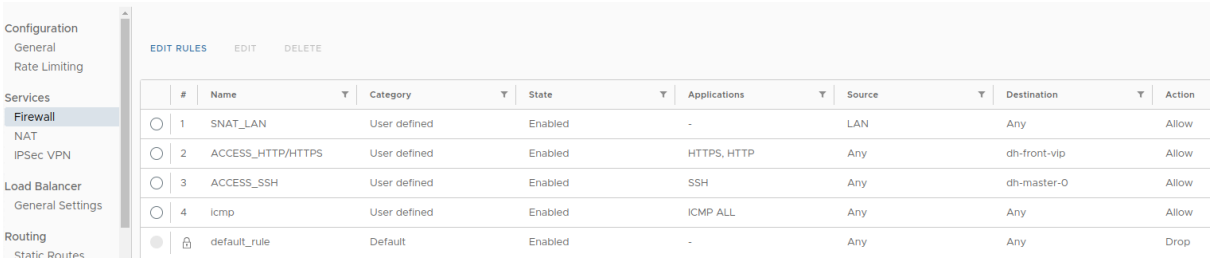
2. Create the following set of IPs (the MetalLB address here is `.10` and the control plane node address is `.2`):


Edit IP Set
Name *
LAN
Description
IP Addresses
Enter an IPv4 or IPv6 address, range or CIDR
ADD
192.168.199.0/24
MODIFY
REMOVE
UNDO
DISCARD
SAVE


Edit IP Set
Name *
dh-front-vip
Description
IP Addresses
Enter an IPv4 or IPv6 address, range or CIDR
ADD
192.168.199.10
MODIFY
REMOVE
UNDO
DISCARD
SAVE



3. Add the following firewall rules:



Virtual machine template

⚠ The provider is confirmed to work with Ubuntu 22.04-based virtual machine templates only.

⚠ Disable vApp/Guest OS Customization (Guest Customization, vApp Customization, and similar mechanisms) for the cluster template and virtual machines in VMware Cloud Director. DP performs the initial node provisioning via `cloud-init` (OVF/VMware GuestInfo datasource). If customization is enabled, it may conflict with `cloud-init` and result in incorrect node initialization.

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

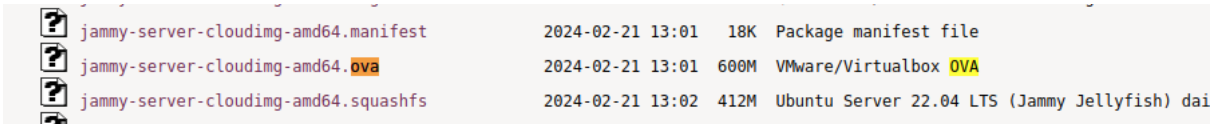
- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

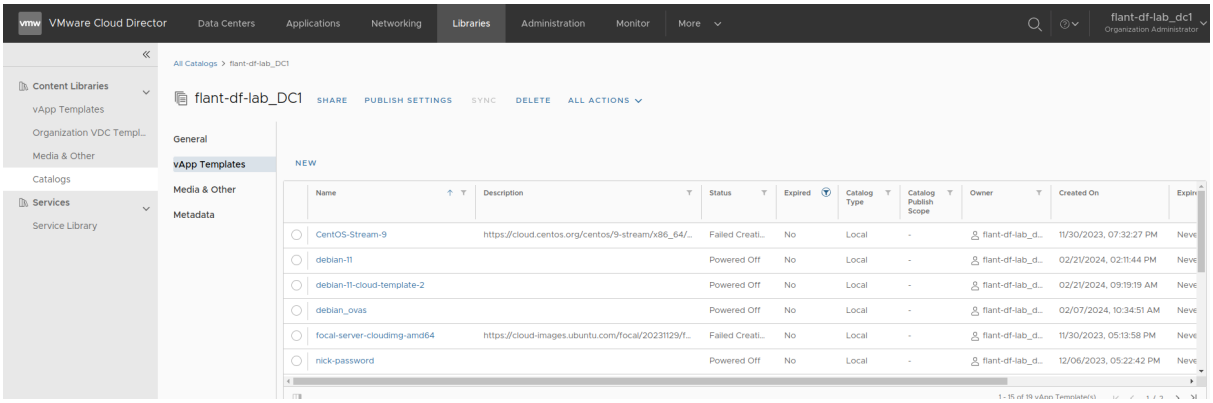
The example below uses the OVA file provided by Ubuntu, updated to include two fixes. Those fixes are essential for CloudPermanent nodes to be provisioned correctly and to be able to mount CSI-created disks.

Making a template from an OVA file

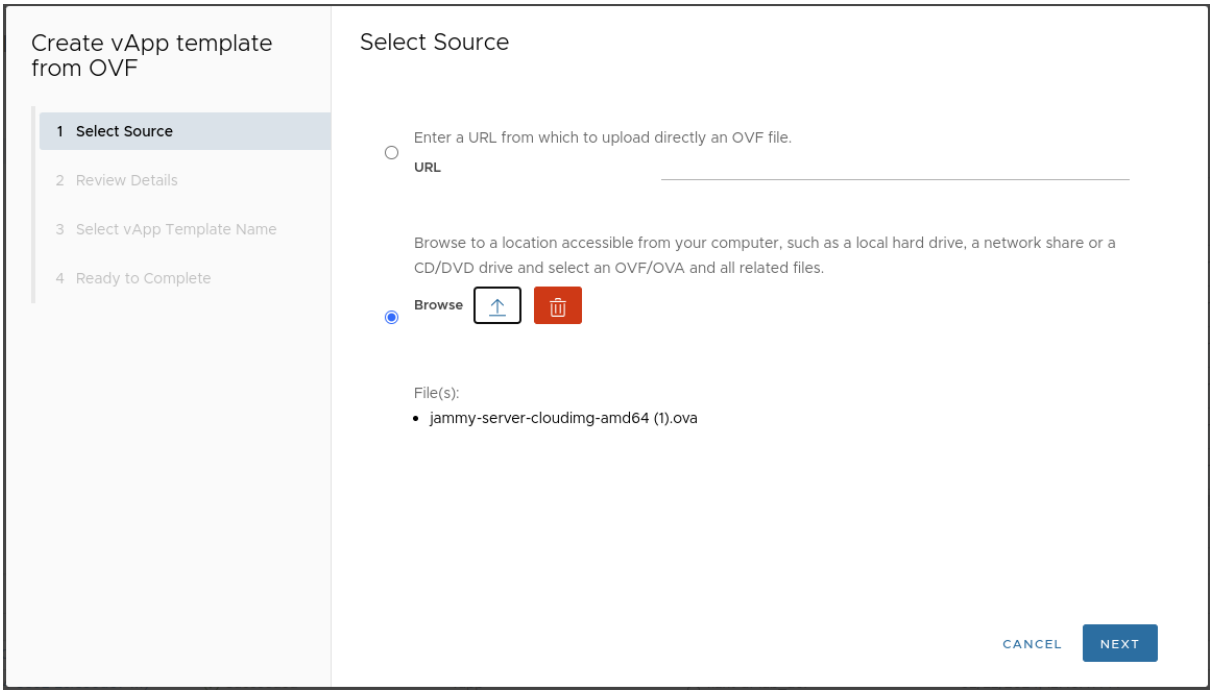
1. Download the OVA file:

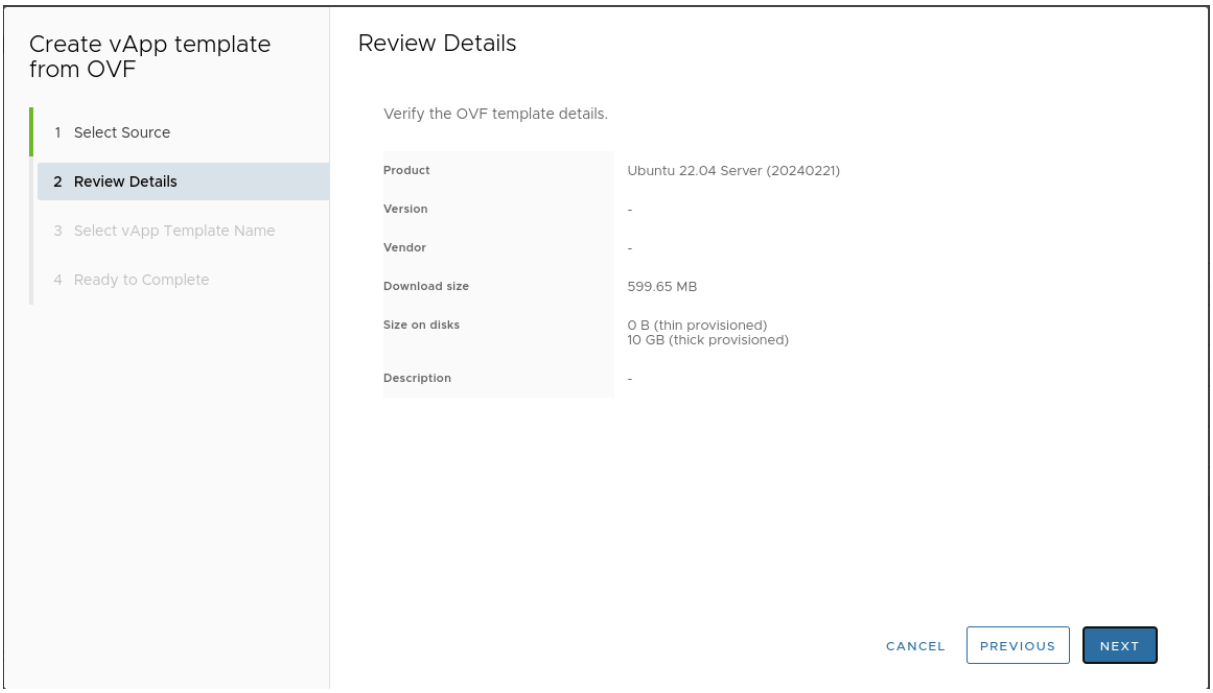


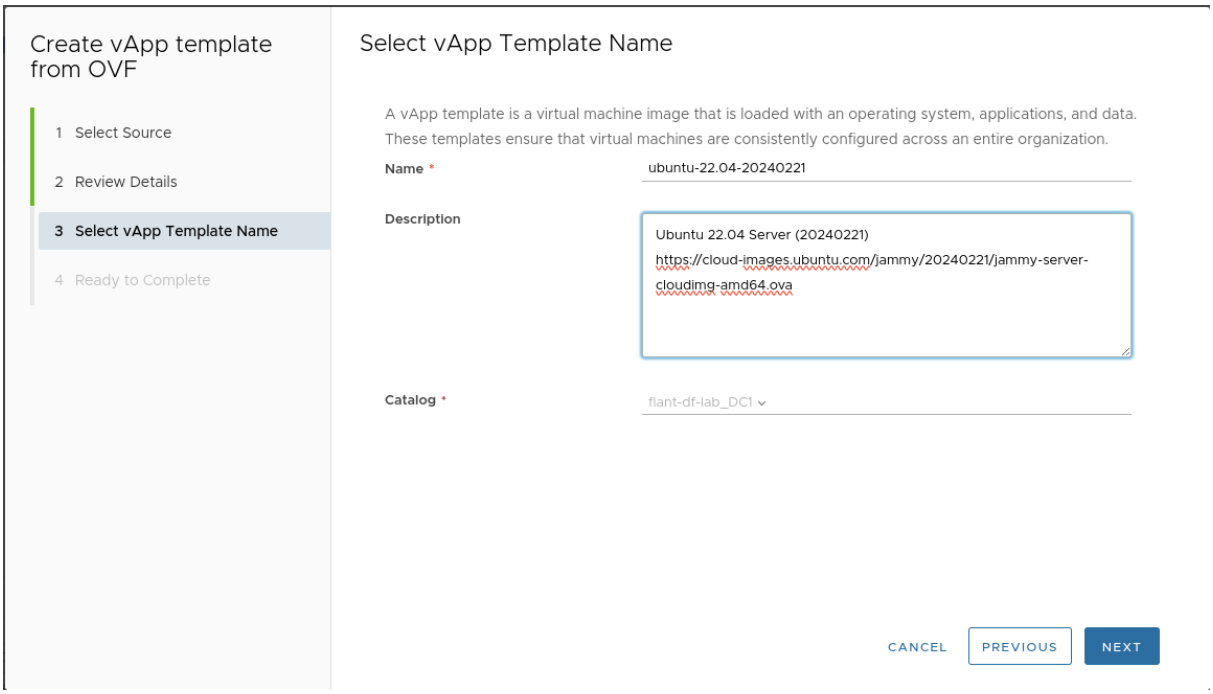
2. Switch to the “Libraries” tab → “Catalogs” → “Organization Catalog”:



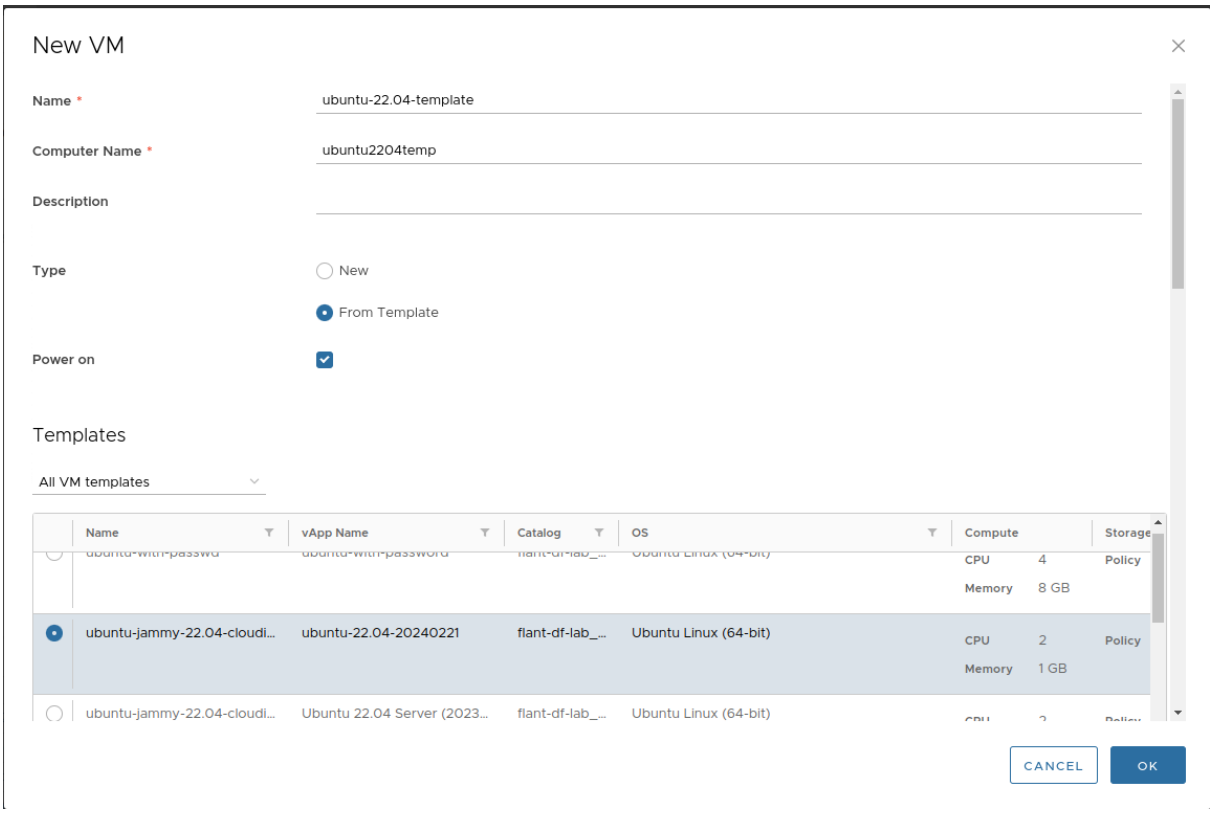
3. Select the template you downloaded and add it to the catalog:

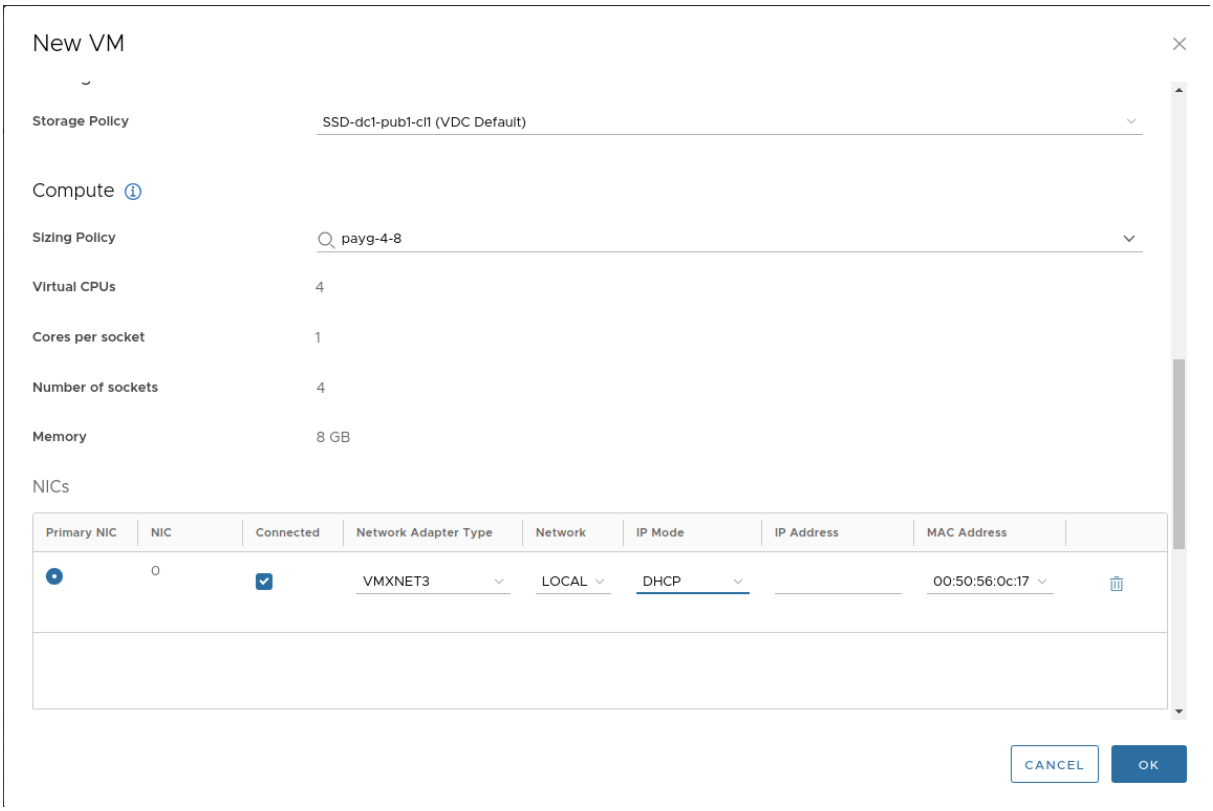




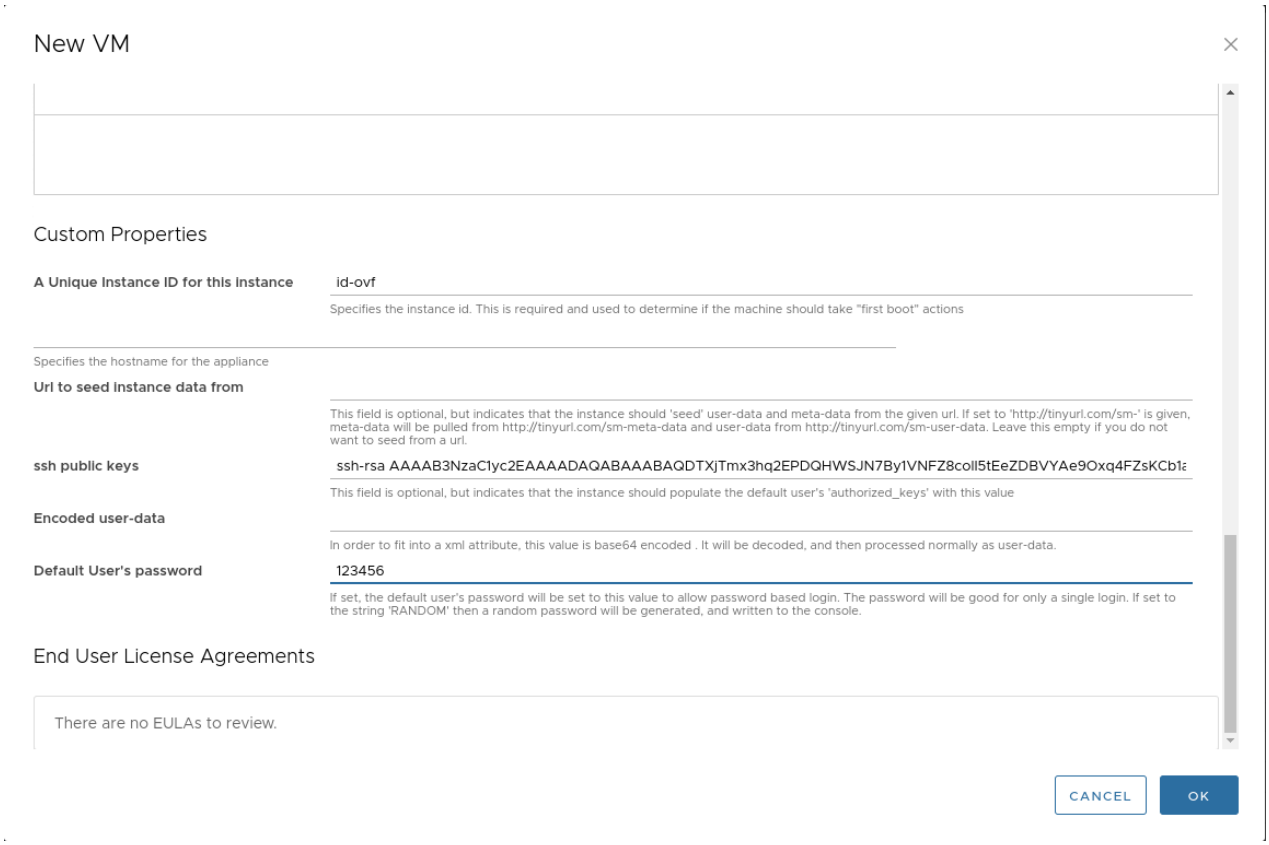


4. Create a virtual machine from the template:



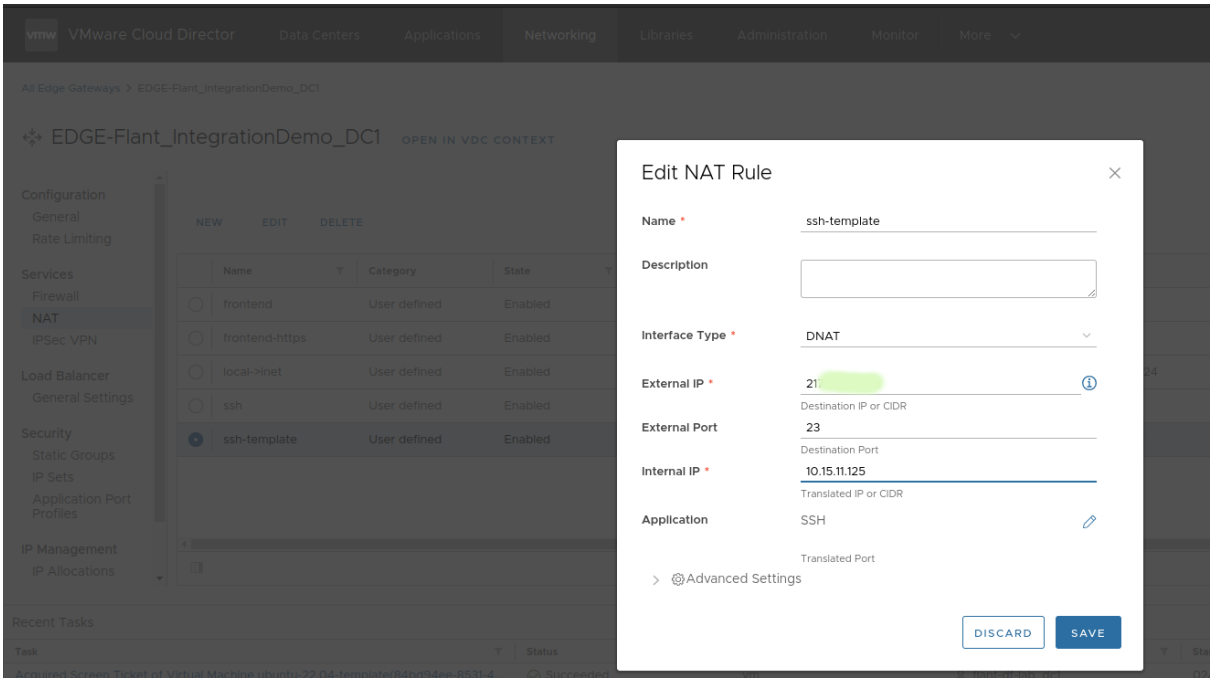


⚠ Enter the default password and public key. You will need them to log in to the VM console.



Follow these steps to be able to connect to the virtual machine:

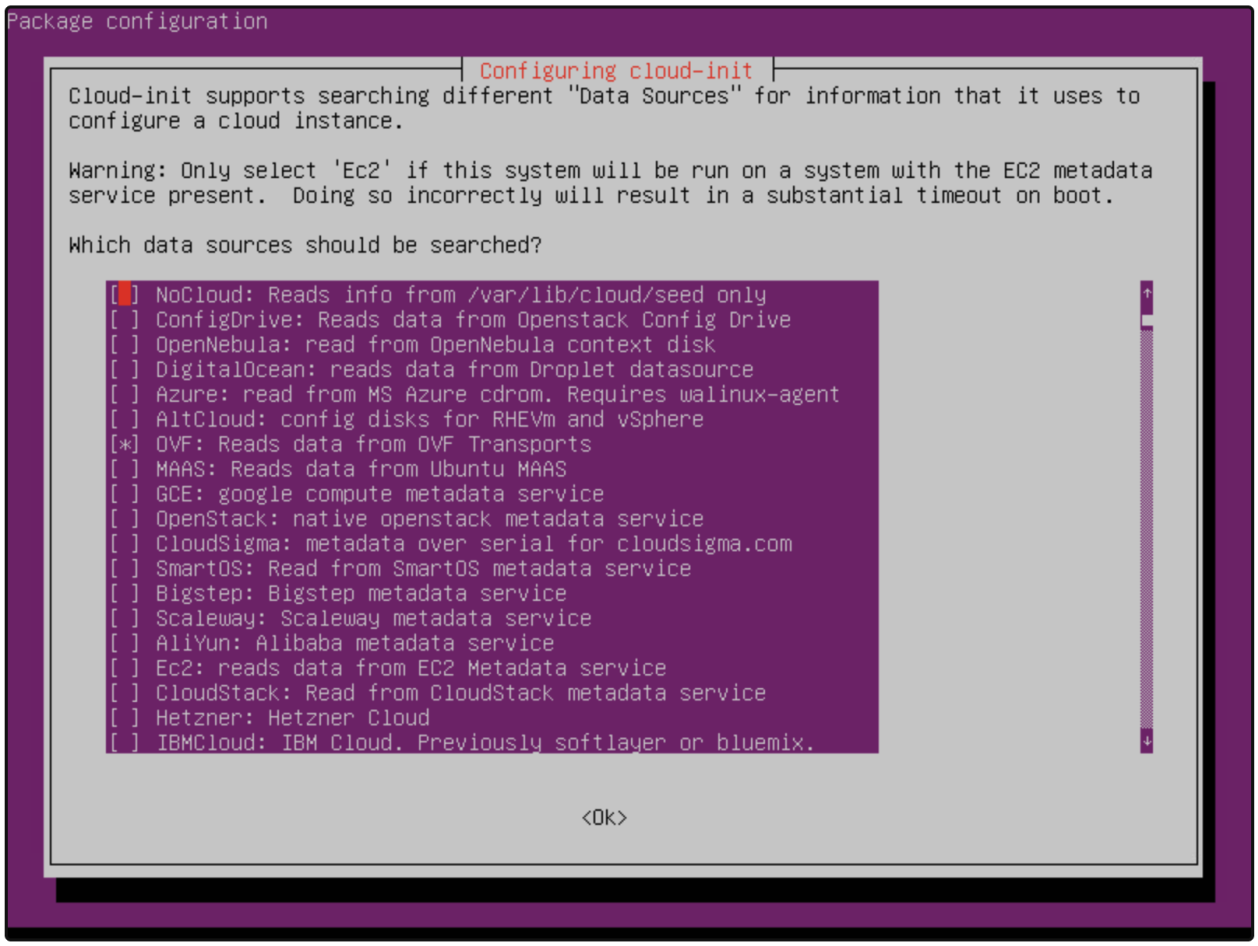
1. Start the virtual machine.
2. Wait for the IP address to be set.
3. Forward port `22` to the virtual machine:



Log on to the virtual machine over SSH and run the following commands:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91_vmware_cust.cfg
dpkg-reconfigure cloud-init
```

In the dialog box that appears, leave the checkmark only on `OVF: Reads data from OVF transports`, and make sure to scroll down and remove checkmarks from other options:



Make sure that the `datasource_list` parameter is specified in the `cloud-init` configuration. You can verify this using the following command:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

If the output is empty, execute the following command:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

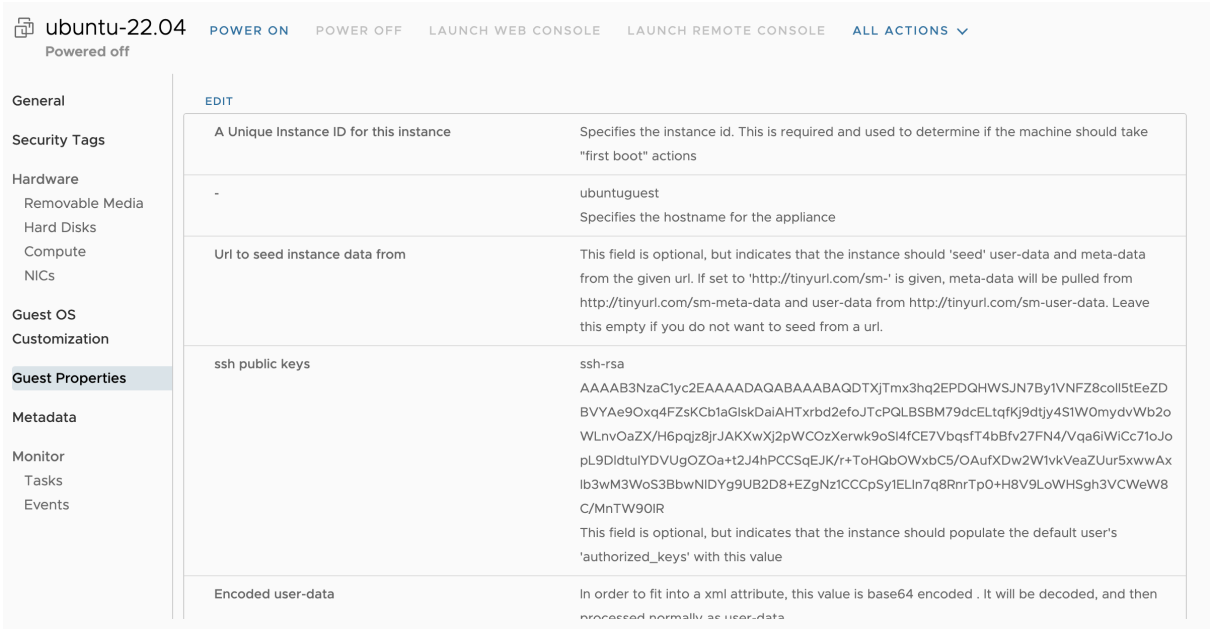
Execute the remaining commands:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

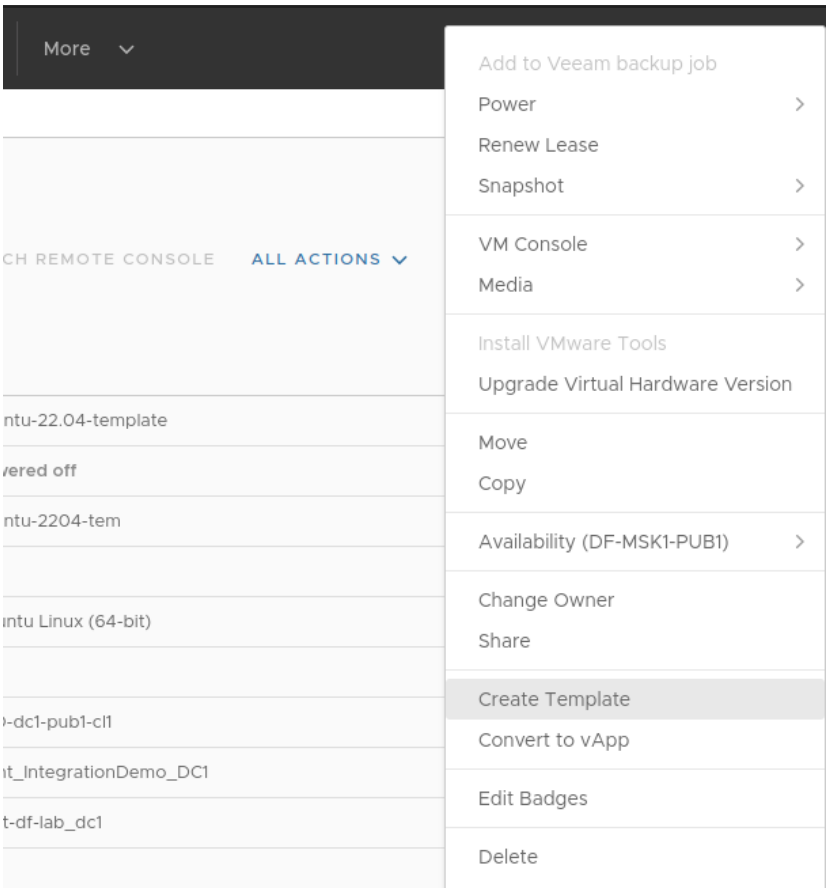
Setting up the template in VCD

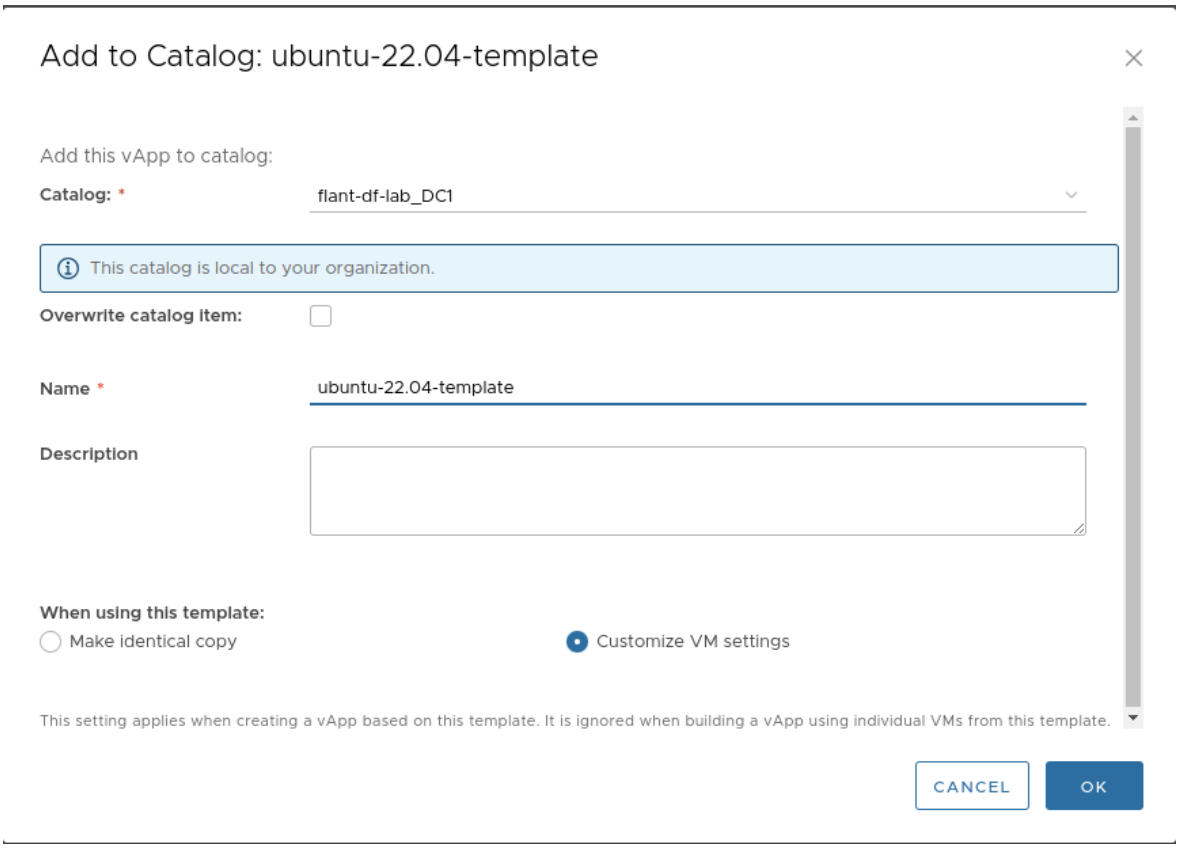
1. Shut down the virtual machine and clear all populated fields in “Guest Properties”:



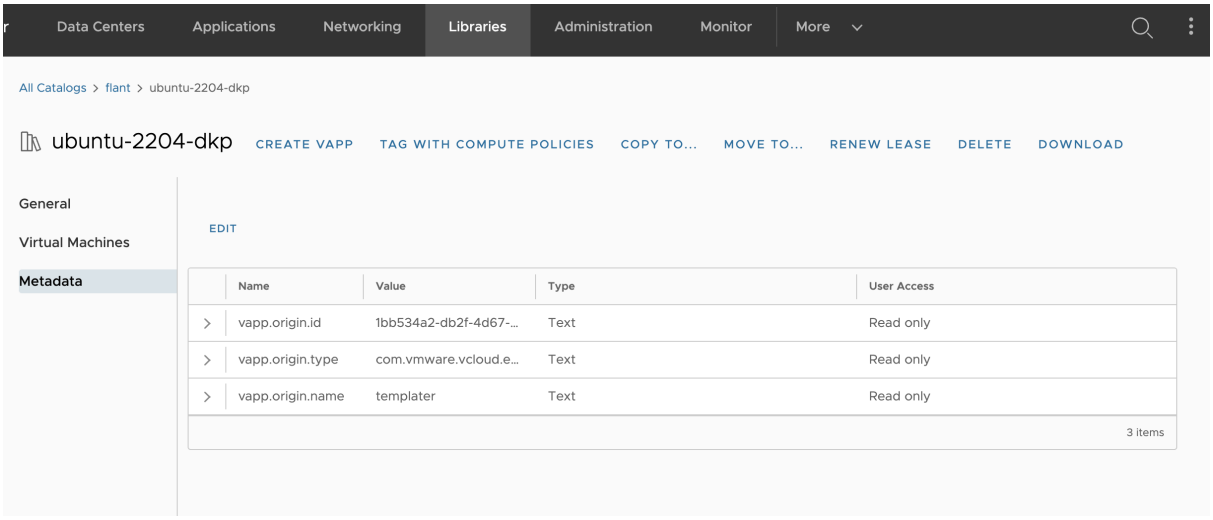


2. Create a virtual machine template:





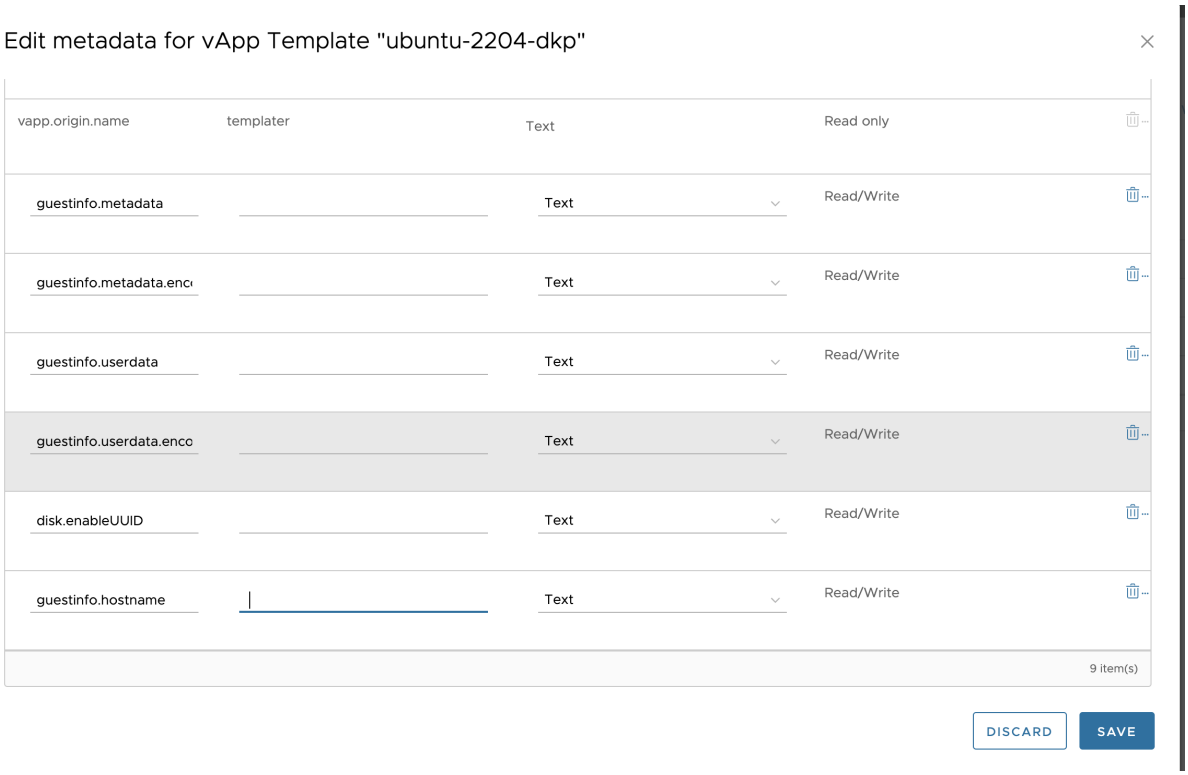
3. In the created template, navigate to the “Metadata” tab and add the following six fields:
   - `guestinfo.metadata`
   - `guestinfo.metadata.encoding`
   - `guestinfo.userdata`
   - `guestinfo.userdata.encoding`
   - `disk.enableUUID`
   - `guestinfo.hostname`



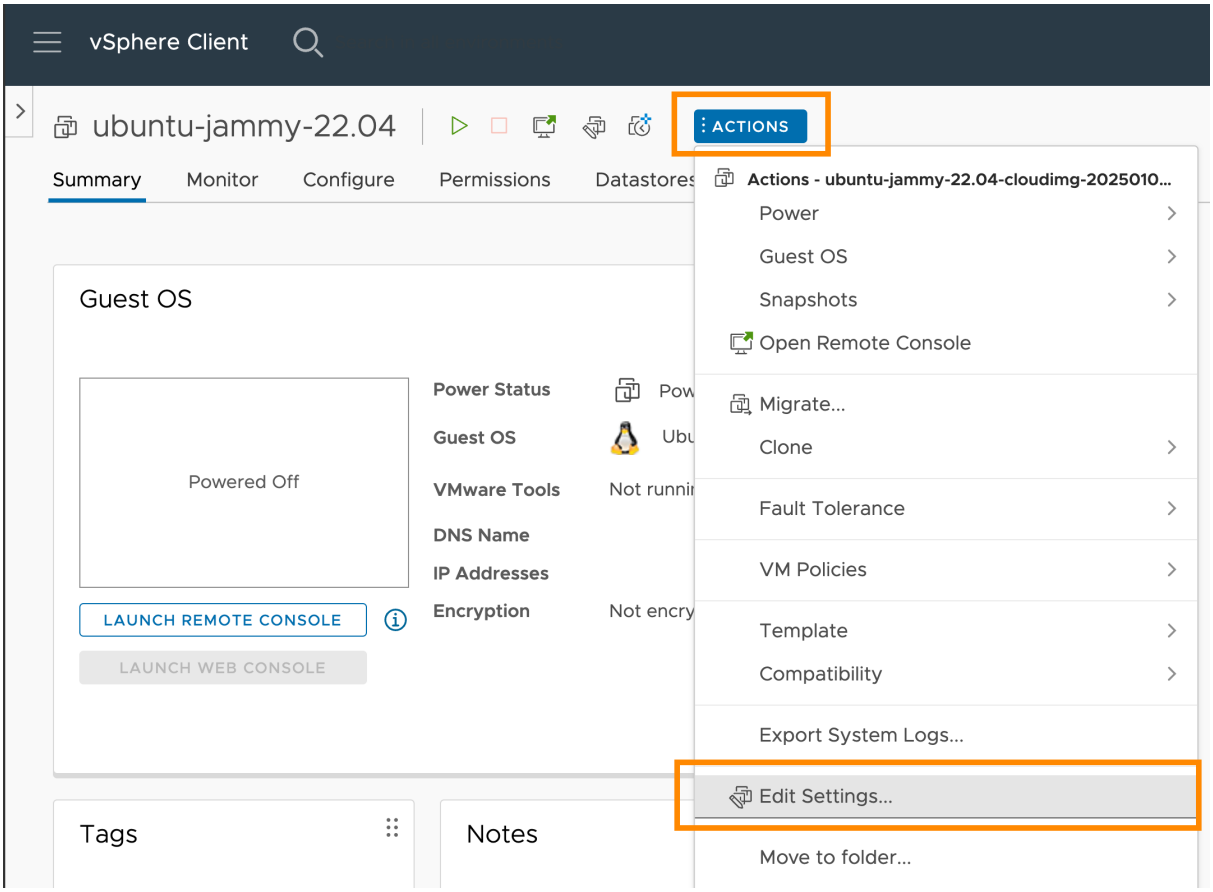
For **each** field in the add/edit form, set the following:

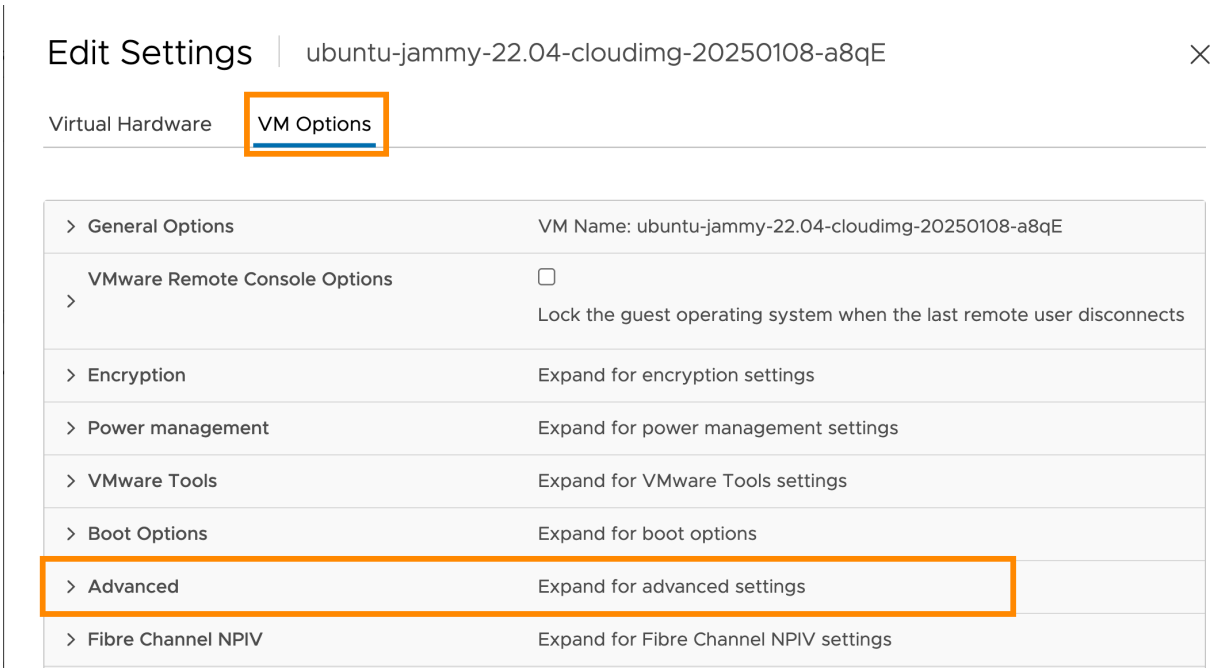
- “Type”: `Text` (text value)
- “User access”: `Read/Write`
- “Value”: A single space

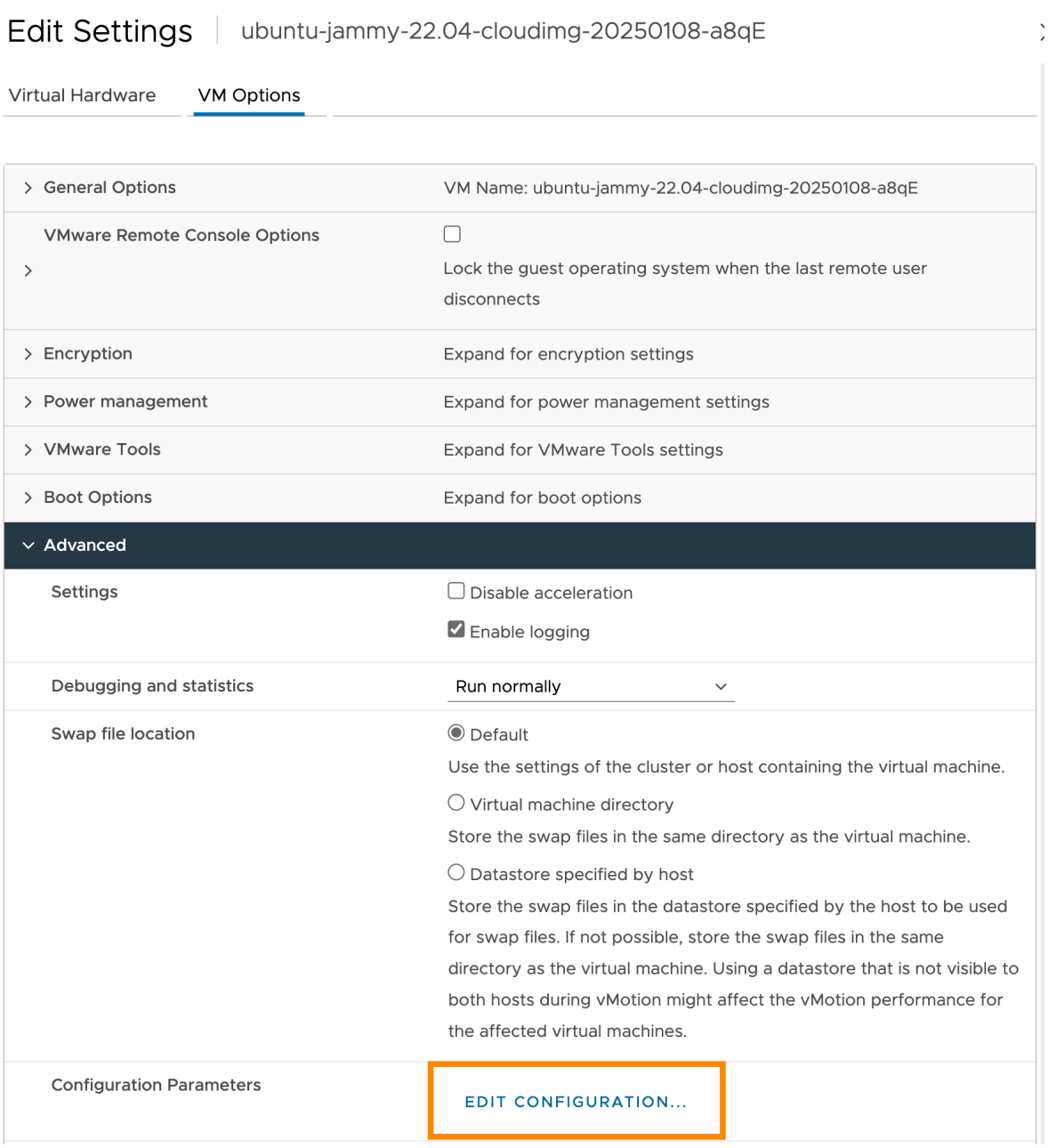
> **i** The VCD UI may not save metadata with an empty value. A single space is used as a placeholder and does not affect functionality. The actual values will be populated automatically when virtual machines are created.

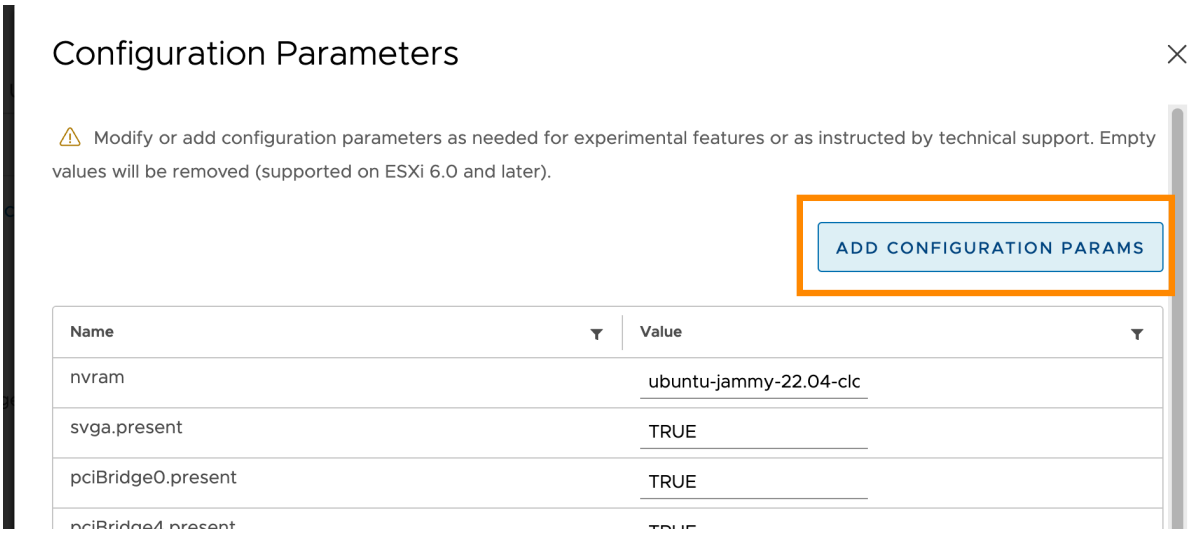


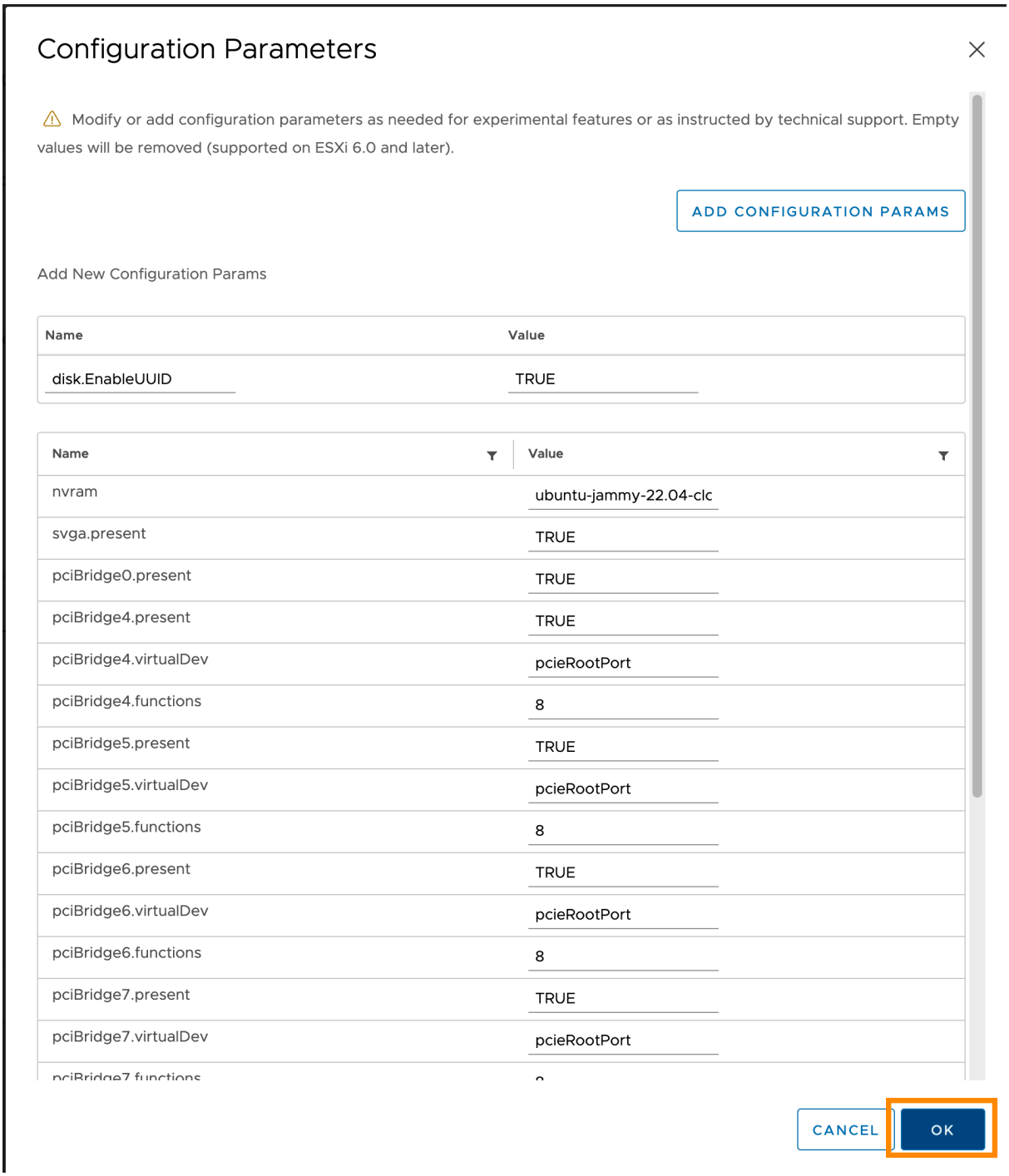
4. In the vCenter management panel for the template, enable the `disk.EnableUUID` parameter:




Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Expand for advanced settings
Fibre Channel NPIV
Expand for Fibre Channel NPIV settings


Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Settings
Disable acceleration
Enable logging
Debugging and statistics
Run normally
Swap file location
Default
Use the settings of the cluster or host containing the virtual machine.
Virtual machine directory
Store the swap files in the same directory as the virtual machine.
Datastore specified by host
Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.
Configuration Parameters
EDIT CONFIGURATION...





Replacing the OS image

Deckhouse Platform creates the nodes from a virtual machine template that is specified in the `template` parameter of the

VCDInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the VCDClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

To move the nodes to a new OS image, prepare a new template based on it and follow the steps from “How do I replace the OS image in a cloud cluster?”.

Using the storage

- VCD supports CSI; disks are created as VCD Independent Disks.
- The `disk.EnableUUID` guest property must be set for the virtual machine templates in use.
- Deckhouse Platform supports disk resizing as of v1.59.1.

Using the LoadBalancer

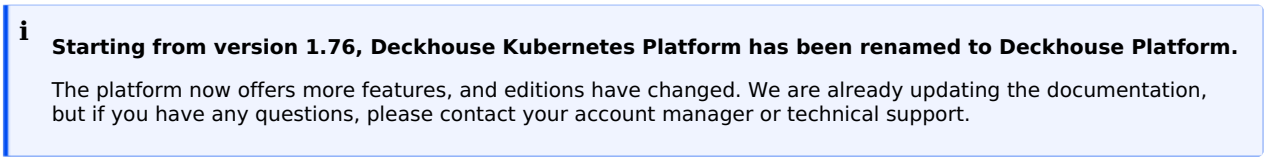
- DP components support `Service` resources of type `LoadBalancer` when deployed on VMware Cloud Director (VCD).
- VMware NSX Advanced Load Balancer (ALB or Avi Networks) is used as the load balancer.
- Support is available **only** when using the `NSX-T` network virtualization platform.
- The load balancer functionality must be enabled on the Edge Gateway by your VCD provider. You can verify this under **Edge Gateway → Load Balancer → General Settings** — the `State` parameter must be `Active`.
- If the load balancer was enabled after the DP cluster was successfully created, the components will automatically pick up the changes within an hour (no additional actions are required).
- For each open port, a **Pool + Virtual Service** pair is created.
- If a firewall is in place, you must create an allow rule for the load balancer’s external IP address and the corresponding ports.

To assign a predefined external IP address to a VCD load balancer, add the `vcd.cpi.flant.com/load-balancer-ip` annotation to the Service.

If both `vcd.cpi.flant.com/load-balancer-ip` annotation and `.spec.loadBalancerIP` filed are set, the annotation takes precedence. The `spec.loadBalancerIP` field remains supported as a deprecated fallback for compatibility.

Example:

```
apiVersion: v1
kind: Service
metadata:
  name: example
  annotations:
    vcd.cpi.flant.com/load-balancer-ip: "192.0.2.10"
spec:
  type: LoadBalancer
  selector:
    app: example
  ports:
    - name: https
      port: 443
      targetPort: 8443
```



Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources



The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.



The module lifecycle stage: General Availability

VCDInstanceClass

Scope: Cluster
Version: v1

Parameters of a group of VCD VirtualMachines used by `cluster-api-controller-manager` (the cloud-provider-vcd module).

- **spec**
  object

  Required value

  - **spec.additionalMetadata**
    object

Custom string key-value pairs to be stored in the Metadata of every node VM, useful for organizing and categorizing your virtualization inventory.

The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the VCDProviderClusterConfiguration spec. This field will take precedence over the existing metadata in cases of duplicate keys.

> **i** **Warning.** Changing the `additionalMetadata` will cause the recreation of CloudEphemeral NodeGroups referring the VCDInstanceClass.

- **spec.affinityRule**
  object

  Specifies the affinity rule applied only to virtual machines of node groups referencing this instance class within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

  - **spec.affinityRule.polarity**
    string

    Required value

    The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

    Allowed values: `Affinity`, `AntiAffinity`

  - **spec.affinityRule.required**
    boolean

    Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

    Default: `false`

- **spec.mainNetwork**
  string

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  Example:

  ```
  mainNetwork: internal
  ```

- **spec.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine. If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **spec.rootDiskSizeGb**
  integer

  root disk size in gigabytes.

  Default: `20`

  Example:

  ```
  rootDiskSizeGb: 20
  ```

- **spec.sizingPolicy**
  string

  SizingPolicy is the sizing policy to be used on this machine. If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

  ```
  sizingPolicy: 4Cpu8Memory
  ```

- **spec.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

```
storageProfile: ssd
```

- **spec.template**
  string

  Required value

  Path of the template OVA that is to be used. Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

  If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

  You can find the template identifier in the URL when you open the template in the vCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

  Examples:

```
template: Linux/ubuntu2204-cloud-ova
```

```
template: MyOrg/Linux/ubuntu2204-cloud-ova
```

```
template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

```
template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Below is an example VCDInstanceClass configuration for ephemeral nodes of VMware Cloud Director cloud provider.

Example configuration of the VCDInstanceClass custom resource

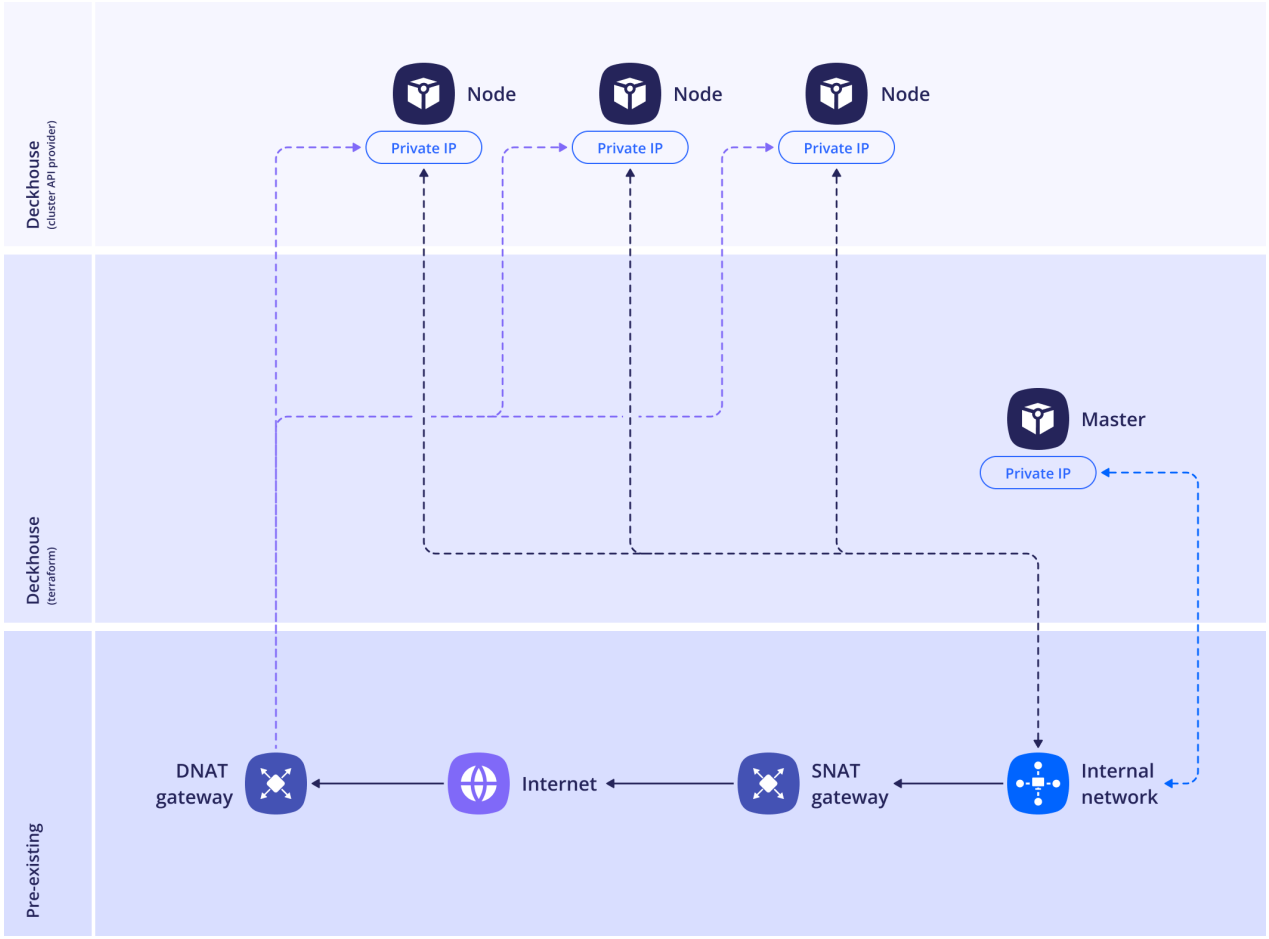
```
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider - VMware Cloud Director: Layouts

> The module lifecycle stage: General Availability

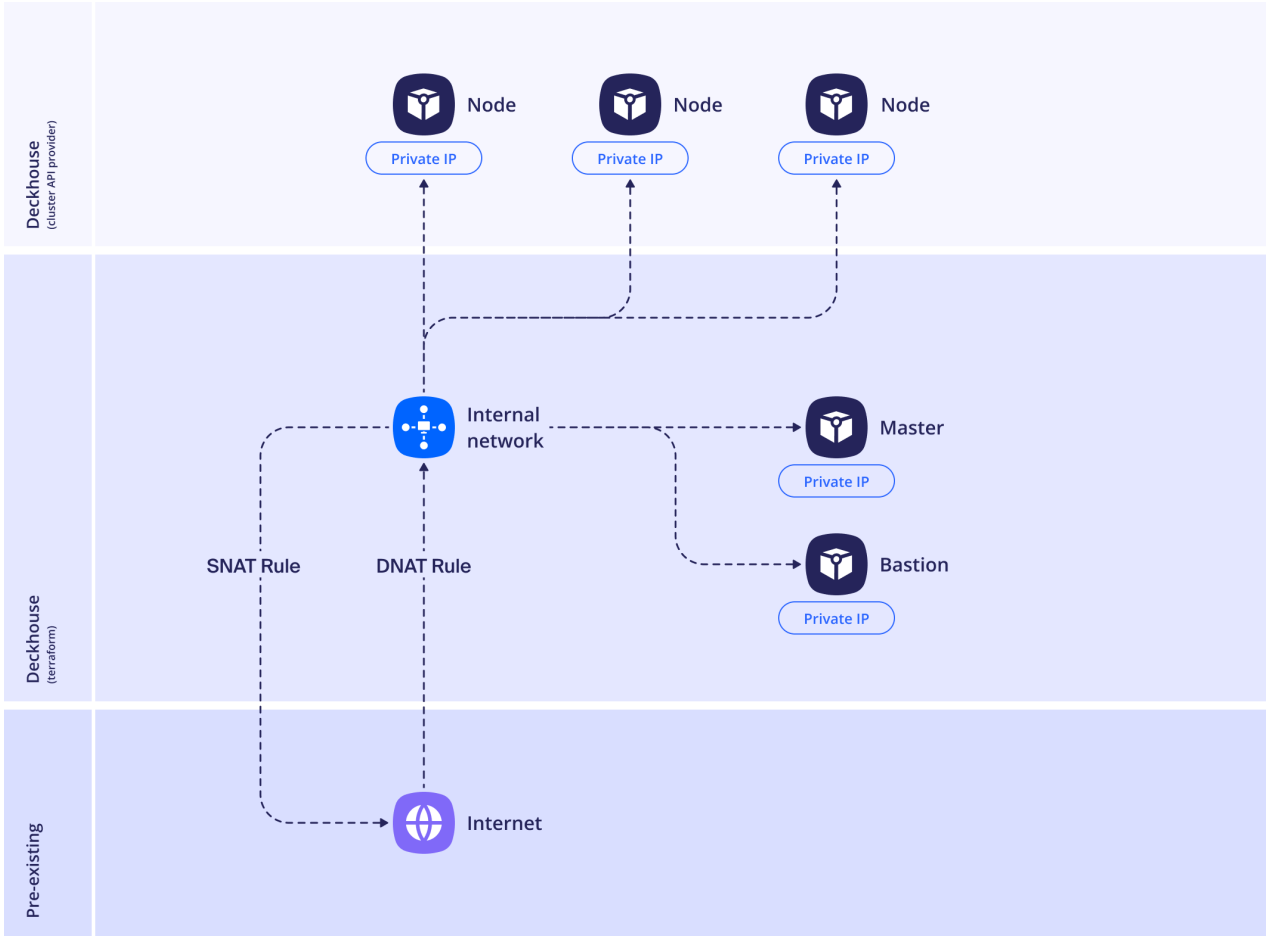
Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

WithNAT



When using this placement scheme, you must check with the administrator which network virtualization platform is in use and specify it in the `edgeGateway.type` parameter.
Two options are supported: `NSX-T` and `NSX-V`.

To ensure administrative access to the cluster nodes, a bastion is deployed. The parameters for its configuration are described in the `bastion` section.

If the Edge Gateway is based on `NSX-T`, a DHCP server will be automatically enabled in the created network for the nodes. It will assign IP addresses starting from the 30th address in the subnet up to the second-to-last (just before the broadcast address).
You can change the starting address of the DHCP pool using the `internalNetworkDHCPPoolStartAddress` parameter.

If `NSX-V` is used, DHCP must be configured manually. Otherwise, nodes that rely on dynamic IP assignment will not be able to obtain an address.

⚠ It is not recommended to use dynamic addressing for the first master node in combination with `NSX-V`.

The deployment scheme assumes automated creation of NAT rules:

- An SNAT rule for translating the addresses of the internal node network to the external address specified in the `edgeGateway.externalIP` property.
- A DNAT rule for translating the external address and port, specified in the `edgeGateway.externalIP` and `edgeGateway.externalPort` properties, respectively, to the internal address of the bastion instance on port 22 using the `TCP` protocol for administrative access to the nodes via SSH.

⚠ If the Edge Gateway is provided by `NSX-V`, you must specify the name and type of the network to which the rule will be bound in the `edgeGateway.NSX-V.externalNetworkName` and `edgeGateway.NSX-V.externalNetworkType` properties, respectively. Typically, this is a network connected to the Edge Gateway in `Gateway Interface` and having an external IP address.

Additionally, you can enable the creation of default firewall rules using the `createDefaultFirewallRules` property.

⚠ If the Edge Gateway is provided by `NSX-T`, existing rules on the Edge Gateway will be overwritten. It is assumed that using this option implies that only one cluster will be deployed per Edge Gateway.

The following rules will be created:

- Allow any outgoing traffic
- Allow incoming traffic over the `TCP` protocol on port 22 to enable SSH access to the cluster nodes
- Allow any incoming traffic over the `ICMP` protocol
- Allow incoming traffic over the `TCP` and `UDP` protocols on ports 30000–32767 for NodePort usage

Example of the layout configuration using `NSX-T`:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Example of the layout configuration using `NSX-V`:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — VMware vSphere

Included in extensions: **On-prem IaaS**

The module lifecycle stage: General Availability

The `cloud-provider-vsphere` module integrates Deckhouse Platform with VMware vSphere. It allows the `node-manager` module to use vSphere resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-vsphere` module:

- Managing vSphere resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the vSphere side;
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in vSphere.
- Provisioning disks via CSI on datastore. By default, CNS volumes with online resize are used. First-Class Disk (FCD) mode is available as legacy and is configured with the `compatibilityFlag` parameter.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-vsphere`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the VsphereInstanceClass resource.
- Registering with `node-manager` so that VsphereInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

⚠ The module is moving CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI). Existing NodeGroups keep running on MCM, new ones are created on CAPI by default. The procedure for moving existing groups is described in How to migrate node groups to Cluster API (CAPI).

Nodes managed by CAPI

vSphere requirement

The vSphere cluster that hosts CAPI-managed nodes must have DRS enabled. The module does not pick a specific host when cloning a virtual machine, so vSphere chooses the host within the cluster. Without DRS the choice is not made, and the nodes of the group are not created.

The same requirement applies to master nodes and CloudPermanent nodes created by Terraform. Node groups managed by MCM also worked in a cluster without DRS.

The remaining environment requirements are listed in Connecting to and authenticating with VMware vSphere.

vCenter tags

The module assigns the `deckhouse-cluster-name/<CLUSTER UUID>` tag to every virtual machine it creates, the same way MCM did. The `deckhouse-node-role/<NODE_GROUP>-<ZONE>` tag that MCM assigned as well is not assigned under CAPI. To select the nodes of a single group, use the `node.deckhouse.io/group` label.

Virtual machine placement

The module honors both placement parameters of the VsphereInstanceClass resource, `resourcePool` and `datastore`. When at least one of them is set in the instance class, the virtual machines of the group are created in the specified resource pool and on the specified datastore in every zone the group spans. A parameter left unset takes the default value defined for the cluster at installation.

Third-party components

List of third-party software used in the `cloud-provider-vsphere` module:

- VMware vSphere Cloud provider v1.33.1, v1.34.0, v1.35.1, v1.35.1, v1.37.0

  License: Apache License 2.0

  Cloud Controller Manager for VMware vSphere.
- VMware vSphere CSI driver v3.7.1, v3.7.1, v3.7.1, v3.7.1, v3.7.3

  License: Apache License 2.0

  CSI driver for VMware vSphere.
- VMware vSphere Terraform provider 2.0.2

  License: Mozilla Public License 2.0

  Terraform provider for VMware vSphere.
- VMware vSphere Cluster API provider v1.15.3

  License: Apache License 2.0

  Infrastructure provider for Kubernetes Cluster API in VMware vSphere.

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

ℹ Included in extensions: **On-prem IaaS**

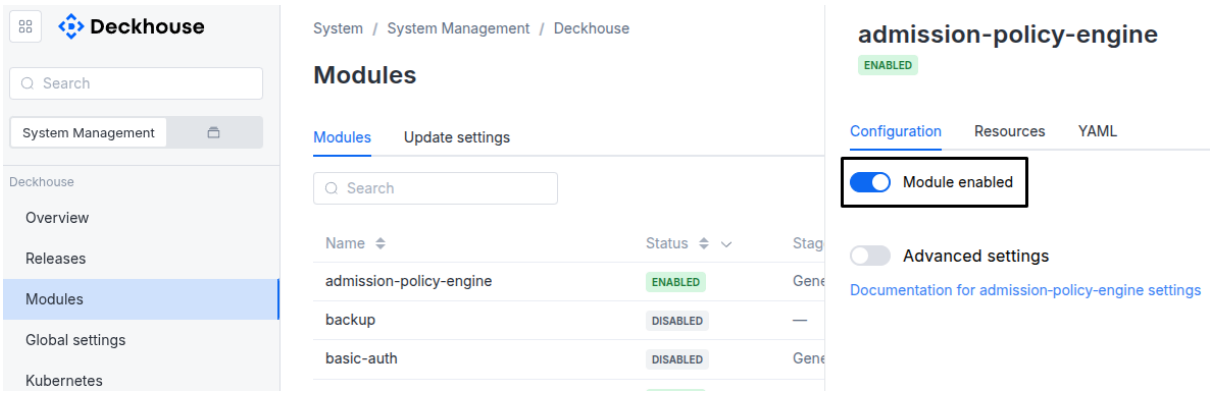
ℹ The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in vSphere.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vsphere` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-vsphere
  ```

- **Using ModuleConfig `cloud-provider-vsphere`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-vsphere` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-vsphere`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-vsphere
  spec:
    enabled: true
  ```

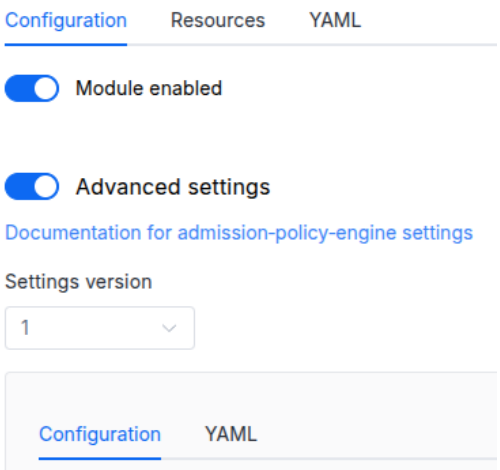
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vsphere` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-vsphere` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-vsphere`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-vsphere` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-vsphere
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the

`spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-vsphere` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  If the field `.storageClass.compatibilityFlag` exists, capitalize the field value.

If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vsphere` module in the Deckhouse configuration (see below). Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VsphereClusterConfiguration structure for configuration.

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the `node-manager` module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the `cloudInstances.ClassReference` parameter of NodeGroup). In the case of the vSphere cloud provider, the instance class is the `VsphereInstanceClass` custom resource that stores specific parameters of the machines.

Storage

The module automatically creates a StorageClass for each Datastore and DatastoreCluster in the zone (or zones).

Also, it can set the name of StorageClass that will be used in the cluster by default (the default parameter), and filter out the unnecessary StorageClasses (the exclude parameter).

CSI

By default, the storage subsystem uses CNS volumes with the ability of online-resize. FCD volumes are also supported, but only in the legacy or migration modes. You can set this via the compatibilityFlag parameter.

Important information concerning the increase of the PVC size

Due to the nature f volume-resizer, CSI, and vSphere API, you have to do the following after increasing the PVC size:

1. On the node where the Pod is located, run the `d8 k cordon <node_name>` command.
2. Delete the Pod.
3. Make sure that the resize was successful. The PVC object must *not have* the `Resizing` condition.

   > **i** The `FileSystemResizePending` state is OK.

4. On the node where the Pod is located, run the `d8 k uncordon <node_name>` command.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.caBundle**
    string

    PEM-encoded CA certificate chain used to verify the vCenter TLS certificate. Use this parameter instead of setting `insecure: true` when the vCenter certificate is signed by a custom or enterprise CA.

  - **settings.disableTimesync**
    boolean

    Disable time synchronization on the vSphere side.

> **Caution!** This parameter will not disable the NTP daemons in the guest OS, but only disable the time correction on the part of ESXi.

- **settings.externalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert ExternalIP into the `.status.addresses` field in the Node API object.

- **settings.host**
  string

  The domain of the vCenter server.

- **settings.insecure**
  boolean

  Set to `true` if vCenter has a self-signed certificate.

- **settings.internalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert InternalIP into the `.status.addresses` field in the Node API object.

- **settings.nsxt**
  object

  Kubernetes load balancer support using NSX-T for the vSphere cloud controller manager.

  - **settings.nsxt.caBundle**
    string

    A certificate authority (CA) certificate chain in PEM format used to verify the NSX-T TLS certificate. Use this parameter instead of `insecureFlag` if the NSX-T TLS certificate is signed by a custom or corporate certificate authority.

  - **settings.nsxt.defaultIpPoolName**
    string

    Required value

    Name of the default IP pool used for the SVC's without `loadbalancer.vmware.io/class` annotation set.

    Example:

    ```
    defaultIpPoolName: pool1
    ```

  - **settings.nsxt.defaultTcpAppProfileName**
    string

    Name of default NSX-T application profile used for TCP connections.

    Default: `default-tcp-lb-app-profile`

    Examples:

    ```
    defaultTcpAppProfileName: default-tcp-lb-app-profile
    ```

    ```
    defaultTcpAppProfileName: tcp-profile1
    ```

  - **settings.nsxt.defaultUdpAppProfileName**
    string

    Name of default NSX-T application profile used for UDP connections.

    Default: `default-udp-lb-app-profile`

    Examples:

    ```
    defaultUdpAppProfileName: default-udp-lb-app-profile
    ```

    ```
    defaultUdpAppProfileName: udp-profile1
    ```

  - **settings.nsxt.host**
    string

    Required value

    NSX-T host.

    Example:

    ```
    host: 1.2.3.4
    ```

- **settings.nsxt.insecureFlag**
  boolean

  To be set to true if NSX-T uses self-signed certificate.

  Examples:

  ```
  insecureFlag: true
  ```

  ```
  insecureFlag: false
  ```

- **settings.nsxt.loadBalancerClass**
  array

  Additional section to define Load Balancer Classes (to use class, set annotation `loadbalancer.vmware.io/class: <class name>` to SVC).

  Examples:

  ```
  loadBalancerClass: []
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
    tcpAppProfileName: profile2
    udpAppProfileName: profile3
  ```

  - **settings.nsxt.loadBalancerClass.ipPoolName**
    string

    Required value

    Name of the IP pool.

  - **settings.nsxt.loadBalancerClass.name**
    string

    Required value

    Load Balancer Class name to use in SVC annotation `loadbalancer.vmware.io/class: <class name>`.

  - **settings.nsxt.loadBalancerClass.tcpAppProfileName**
    string

    Name of application profile used for TCP connections.

    Default: `defaultTcpAppProfileName`

  - **settings.nsxt.loadBalancerClass.udpAppProfileName**
    string

    Name of application profile used for UDP connections.

    Default: `defaultUdpAppProfileName`

- **settings.nsxt.password**
  string

  Required value

  NSX-T password.

  Example:

  ```
  password: password
  ```

- **settings.nsxt.size**
  string

  Size of load balancer service.

  Default: `MEDIUM`

  Allowed values: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

  Example:

  ```
  size: SMALL
  ```

- **settings.nsxt.tier1GatewayPath**
  string

  Required value

  Policy path for the NSX-T tier1 gateway.

  Example:

  ```
  tier1GatewayPath: "/path/tier1"
  ```

- **settings.nsxt.user**
  string

  Required value

  NSX-T user name.

  Example:

  ```
  user: user
  ```

- **settings.password**
  string

  The user's password.

- **settings.region**
  string

  Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

- **settings.regionTagCategory**
  string

  The name of the tag category used to identify the region (vSphere Datacenter).

- **settings.sshKeys**
  array of strings

  A list of public SSH keys in plain-text format.

- **settings.storageClass**
  object

  - **settings.storageClass.compatibilityFlag**
    string

    A flag allowing the use of the old CSI version:

    - `Legacy` — use the old version of the driver. FCD discs only, no online-resizing;
    - `Migration` — in this case, both drivers will be available in the cluster at the same time. This mode is used to migrate from an old driver.

    Allowed values: `Legacy`, `Migration`

  - **settings.storageClass.default**
    Deprecated
    string

    The name of StorageClass that will be used in the cluster by default.

    If the parameter is omitted, the default StorageClass is either:

    - an arbitrary StorageClass present in the cluster that has the default annotation;
    - the first (in lexicographic order) StorageClass created by the module.

    > **i** **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

    Example:

    ```
    default: fast-lun102-7d0bf578
    ```

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-lun101-.*"
    - slow-lun103-1c280603
    ```

- **settings.username**
  string

Username.

> **Warning.** Make sure to specify the username together with the domain, for example: `username@domain.local`.
>
> You can find the correct domain in the vCenter control panel.

- **settings.vmFolderPath**
  string

  The path to the VirtualMachine Folder where the cloned VMs will be created.

- **settings.zoneTagCategory**
  string

  The name of the tag category used to identify the zone (vSphere Cluster).

- **settings.zones**
  array of strings

  The globally restricted set of zones that this Cloud Provider works with.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

> If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vsphere` module in the Deckhouse configuration. Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VsphereClusterConfiguration structure for configuration.
>
> Additional info about Vsphere Cloud Load Balancers.

VsphereClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in vSphere.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes section. Information about reducing the number of nodes and deleting nodes is provided in the Integration with VMware vSphere services section.

Example:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
layout: Standard
vmFolderPath: folder/prefix
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: region2
zones:
- region2-a
externalNetworkNames:
- net3-k8s
internalNetworkNames:
- K8S_3
internalNetworkCIDR: 172.16.2.0/24
baseResourcePool: kubernetes/cloud
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    mainNetwork: net3-k8s
    additionalNetworks:
    - K8S_3
    datastore: lun10
    rootDiskSize: 50
    runtimeOptions:
      nestedHardwareVirtualization: false
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    datastore: lun10
    mainNetwork: net3-k8s
provider:
  server: "<SERVER>"
  username: "<USERNAME@DOMAIN.LOCAL>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **baseResourcePool**
  string

  A path (relative to vSphere Cluster) to the existing parent `resourcePool` for all `resourcePool` created in each zone.

- **disableTimesync**
  boolean

  Disable time synchronization on the vSphere side.

  > **Caution!** This parameter will not disable the NTP daemons in the guest OS, but only disable the time correction on the part of ESXi.

  Default: `true`

- **externalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert ExternalIP into the `.status.addresses` field in the Node API object.

  Example:

```
externalNetworkNames:
- MAIN-1
- public
```

- **internalNetworkCIDR**
  string

  Subnet for master nodes in the internal network.

  Addresses are allocated starting with the tenth address. E.g., if you have the `192.168.199.0/24` subnet, addresses will be allocated starting with `192.168.199.10`.

  The `internalNetworkCIDR` is used if `additionalNetworks` are defined in `masterInstanceClass`.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert InternalIP into the `.status.addresses` field in the Node API object.

  Example:

```
internalNetworkNames:
- KUBE-3
- devops-internal
```

- **kind**
  string

  Required value

  Allowed values: `VsphereClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the VsphereInstanceClass.

    - **masterNodeGroup.instanceClass.additionalNetworks**
      array of strings

      Paths to networks that VirtualMachines' secondary NICs will connect to. Relative to the datacenter.

      Example:

```
additionalNetworks:
- DEVOPS_32
- DEVOPS_50
```

    - **masterNodeGroup.instanceClass.datastore**
      string

      Required value

      Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

      Both a plain Datastore name (`lun-1201`) and a path with parent folders (`DS_PARENT_FOLDER/lun-1201`) are supported.

      Example:

```
datastore: lun-1201
```

- **masterNodeGroup.instanceClass.mainNetwork**
  string

  Required value

  Path to the network that VirtualMachines’ primary NICs will connect to (default gateway). Relative to the datacenter.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **masterNodeGroup.instanceClass.mainNetworkIPAddresses**
  array of objects

  A list of static IP addresses with a CIDR prefix for the main network specified in the `mainNetwork` parameter. Deckhouse Platform assigns addresses to nodes in list order and does not return to the beginning of the list after the last address.

  For a non-empty list, provide at least one address for each node in the group.

  If the list contains fewer addresses than nodes in the group, infrastructure planning fails.

  If the list is empty or omitted, nodes obtain their IP addresses through DHCP.

  Example:

  ```
  mainNetworkIPAddresses:
  - address: 10.1.14.20/24
    gateway: 10.1.14.254
    nameservers:
      addresses:
      - 8.8.8.8
      - 8.8.4.4
  ```

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.address**
    string

    Required value

    An IP address with a CIDR prefix.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.gateway**
    string

    Required value

    The IP address of the default gateway.

    It must be located in the subnet specified in the `address` parameter

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers**
    object

    Required value

    - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.addresses**
      array of strings

      A list of DNS servers.

      Example:

      ```
      addresses:
      - 8.8.8.8
      - 8.8.4.4
      ```

      - **Element of the array**
        string

        Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

    - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.search**
      array of strings

      A list of DNS search domains.

      Example:

```
search:
- tech.lan
```

- **masterNodeGroup.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to vSphere VirtualMachines.

  Example:

  ```
  memory: 8192
  ```

- **masterNodeGroup.instanceClass.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **masterNodeGroup.instanceClass.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  Example:

  ```
  resourcePool: rp-2012
  ```

- **masterNodeGroup.instanceClass.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Example:

  ```
  rootDiskSize: 50
  ```

- **masterNodeGroup.instanceClass.runtimeOptions**
  object

  Additional VM’s parameters.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuReservation**
    integer

    CPU reservation in MHz.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuShares**
    integer

    The relative amount of CPU Shares for VMs to be created.

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryLimit**
    integer

    Memory limit in MB.

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryReservation**
    integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Allowed values: `0 <= X <= 100`

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryShares**
    integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **masterNodeGroup.instanceClass.runtimeOptions.nestedHardwareVirtualization**
    boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

        - **masterNodeGroup.instanceClass.template**
          string

          Required value

          Path to the template to be cloned. Relative to the datacenter.

          Example:

          ```
          template: dev/golden_image
          ```

      - **masterNodeGroup.replicas**
        integer

        Required value

        The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

        Allowed values: `1 <= X`

      - **masterNodeGroup.zones**
        array of strings

        A limited set of zones in which master nodes can be created.

        The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the VsphereInstanceClass.

    - **nodeGroups.instanceClass.additionalNetworks**
      array of strings

      Paths to networks that VirtualMachines’ secondary NICs will connect to. Relative to the datacenter.

      Example:

      ```
      additionalNetworks:
      - DEVOPS_32
      - DEVOPS_50
      ```

    - **nodeGroups.instanceClass.datastore**
      string

      Required value

      Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

      Both a plain Datastore name (`lun-1201`) and a path with parent folders (`DS_PARENT_FOLDER/lun-1201`) are supported.

      Example:

      ```
      datastore: lun-1201
      ```

    - **nodeGroups.instanceClass.mainNetwork**
      string

      Required value

      Path to the network that VirtualMachines’ primary NICs will connect to (default gateway). Relative to the datacenter.

      Example:

      ```
      mainNetwork: k8s-msk-178
      ```

    - **nodeGroups.instanceClass.mainNetworkIPAddresses**
      array of objects

      A list of static IP addresses with a CIDR prefix for the main network specified in the `mainNetwork` parameter. Deckhouse Platform assigns addresses to nodes in list order and does not return to the beginning of the list after the last address.

      For a non-empty list, provide at least one address for each node in the group.

If the list contains fewer addresses than nodes in the group, infrastructure planning fails.

If the list is empty or omitted, nodes obtain their IP addresses through DHCP.

Example:

```
mainNetworkIPAddresses:
- address: 10.1.14.20/24
  gateway: 10.1.14.254
  nameservers:
    addresses:
    - 8.8.8.8
    - 8.8.4.4
```

- **nodeGroups.instanceClass.mainNetworkIPAddresses.address**
  string

  Required value

  An IP address with a CIDR prefix.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **nodeGroups.instanceClass.mainNetworkIPAddresses.gateway**
  string

  Required value

  The IP address of the default gateway.

  It must be located in the subnet specified in the `address` parameter

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers**
  object

  Required value

  - **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.addresses**
    array of strings

    A list of DNS servers.

    Example:

    ```
    addresses:
    - 8.8.8.8
    - 8.8.4.4
    ```

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.search**
    array of strings

    A list of DNS search domains.

    Example:

    ```
    search:
    - tech.lan
    ```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to vSphere VirtualMachines.

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

```
numCPUs: 2
```

- **nodeGroups.instanceClass.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  Example:

  ```
  resourcePool: rp-2012
  ```

- **nodeGroups.instanceClass.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Example:

  ```
  rootDiskSize: 50
  ```

- **nodeGroups.instanceClass.runtimeOptions**
  object

  Additional VM’s parameters.

  - **nodeGroups.instanceClass.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **nodeGroups.instanceClass.runtimeOptions.cpuReservation**
    integer

    CPU reservation in MHz.

  - **nodeGroups.instanceClass.runtimeOptions.cpuShares**
    integer

    The relative amount of CPU Shares for VMs to be created.

  - **nodeGroups.instanceClass.runtimeOptions.memoryLimit**
    integer

    Memory limit in MB.

  - **nodeGroups.instanceClass.runtimeOptions.memoryReservation**
    integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Allowed values: `0 <= X <= 100`

  - **nodeGroups.instanceClass.runtimeOptions.memoryShares**
    integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **nodeGroups.instanceClass.runtimeOptions.nestedHardwareVirtualization**
    boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

- **nodeGroups.instanceClass.template**
  string

  Required value

  Path to the template to be cloned. Relative to the datacenter.

  Example:

  ```
  template: dev/golden_image
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**

object

The same as the `metadata.annotations` standard field.

Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**
  object

  A list of labels to attach to cluster resources.

  The same as the `metadata.labels` standard field.

  Example:

  ```
  labels:
    environment: production
    app: warp-drive-ai
  ```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeGroups.zones**
  array of strings

  A limited set of zones in which nodes can be created.

- **nsxt**
  object

  Kubernetes load balancer support using NSX-T for the vSphere cloud controller manager.

  - **nsxt.caBundle**
    string

    A certificate authority (CA) certificate chain in PEM format used to verify the NSX-T TLS certificate. Use this parameter instead of `insecureFlag` if the NSX-T TLS certificate is signed by a custom or corporate certificate authority.

  - **nsxt.defaultIpPoolName**
    string

    Required value

    Name of the default IP pool used for the SVC's without `loadbalancer.vmware.io/class` annotation set.

    Example:

    ```
    defaultIpPoolName: pool1
    ```

  - **nsxt.defaultTcpAppProfileName**
    string

    Name of default NSX-T application profile used for TCP connections.

Default: `default-tcp-lb-app-profile`

Examples:

```
defaultTcpAppProfileName: default-tcp-lb-app-profile
```

```
defaultTcpAppProfileName: tcp-profile1
```

- **nsxt.defaultUdpAppProfileName**
  string

  Name of default NSX-T application profile used for UDP connections.

  Default: `default-udp-lb-app-profile`

  Examples:

  ```
  defaultUdpAppProfileName: default-udp-lb-app-profile
  ```

  ```
  defaultUdpAppProfileName: udp-profile1
  ```

- **nsxt.host**
  string

  Required value

  NSX-T host.

  Example:

  ```
  host: 1.2.3.4
  ```

- **nsxt.insecureFlag**
  boolean

  To be set to true if NSX-T uses self-signed certificate.

  Examples:

  ```
  insecureFlag: true
  ```

  ```
  insecureFlag: false
  ```

- **nsxt.loadBalancerClass**
  array

  Additional section to define Load Balancer Classes (set annotation `loadbalancer.vmware.io/class: <CLASS NAME>` to SVC to use the class).

  Examples:

  ```
  loadBalancerClass: []
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
    tcpAppProfileName: profile2
    udpAppProfileName: profile3
  ```

  - **nsxt.loadBalancerClass.ipPoolName**
    string

    Required value

    Name of the IP pool.

  - **nsxt.loadBalancerClass.name**
    string

    Required value

    Load Balancer Class name to use in SVC annotation `loadbalancer.vmware.io/class: <CLASS NAME>`.

  - **nsxt.loadBalancerClass.tcpAppProfileName**
    string

    Name of application profile used for TCP connections.

Default: `defaultTcpAppProfileName`

- **nsxt.loadBalancerClass.udpAppProfileName**
  string

  Name of application profile used for UDP connections.

  Default: `defaultUdpAppProfileName`

- **nsxt.password**
  string

  Required value

  NSX-T password.

  Example:

  ```
  password: password
  ```

- **nsxt.size**
  string

  Size of load balancer service.

  Default: `MEDIUM`

  Allowed values: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

  Example:

  ```
  size: SMALL
  ```

- **nsxt.tier1GatewayPath**
  string

  Required value

  Policy path for the NSX-T tier1 gateway.

  Example:

  ```
  tier1GatewayPath: "/path/tier1"
  ```

- **nsxt.user**
  string

  Required value

  NSX-T user name.

  Example:

  ```
  user: user
  ```

- **provider**
  object

  Required value

  Parameters for connecting to the vCenter.

  - **provider.caBundle**
    string

    PEM-encoded CA certificate chain used to verify the vCenter TLS certificate. Use this parameter instead of setting `insecure: true` when the vCenter certificate is signed by a custom or enterprise CA.

  - **provider.insecure**
    boolean

    Set to `true` if vCenter has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    Required value

    The user’s password.

  - **provider.server**
    string

    Required value

    The host or the IP address of the vCenter server.

  - **provider.username**

string

Required value

Username.

> **Warning.** Make sure to specify the username together with the domain, for example: `username@domain.local`.
>
> You can find the correct domain in the vCenter control panel.

- **region**
  string

  Required value

  Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

- **regionTagCategory**
  string

  Required value

  The name of the tag category used to identify the region (vSphere Datacenter).

  Default: `k8s-region`

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **storagePolicyID**
  string

  ID of the SPBM (Storage Policy Based Management) storage policy.

- **useNestedResourcePool**
  boolean

  Create nested resource pool (`true`) or use main resource pool (`false`).

  Default: `true`

- **vmFolderExists**
  boolean

  Set the value to `true` if the path specified in `vmFolderPath` exists. Installing more than one cluster in a folder is not possible.

  Default: `false`

- **vmFolderPath**
  string

  Required value

  The path to the VirtualMachine Folder where the cloned VMs will be created.

  Example:

  ```
  vmFolderPath: dev/test
  ```

- **zoneTagCategory**
  string

  Required value

  The name of the tag category used to identify the zone (vSphere Cluster).

  Default: `k8s-zone`

- **zones**
  array of strings

  Required value

  The globally restricted set of zones that this Cloud Provider works with.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> **i** The module lifecycle stage: General Availability

Environment requirements

The following prerequisites must be met for Deckhouse Platform to work correctly with VMware vSphere:

- Access to vCenter;
- A user account with the required set of privileges;
- Tags and tag categories created in vSphere;
- Networks with DHCP and Internet access;
- Shared Datastore resources available on all ESXi hosts in use;
- vSphere version `7.x` or `8.x` with support for `Online volume expansion`;
- vCenter accessible from inside the cluster from the master nodes;
- A configured Datacenter that includes the following objects:
  1. VirtualMachine template:
     - The virtual machine image must use `Virtual machines with hardware version 15 or later` — this is required for online resize support.
     - The image must include the `open-vm-tools`, `cloud-init`, and `cloud-init-vmware-guestinfo` packages if the `cloud-init` version is lower than `21.3`.
  2. Network:
     - The network must be available on all ESXi hosts where virtual machines are planned to be created.
  3. Datastore (one or more):
     - The Datastore must be connected to all ESXi hosts where virtual machines are planned to be created.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Datastore (default: `k8s-zone`). This tag defines the zone.
     - All Cluster objects within the same zone must have access to all Datastore objects in that zone.
  4. Cluster:
     - All ESXi hosts in use must be added to the Cluster.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Cluster (default: `k8s-zone`). This tag defines the zone.
     - DRS must be enabled in the Cluster. The platform does not pick a specific host when creating a virtual machine, so vSphere chooses the host.
  5. Folder for the virtual machines being created:
     - This parameter is optional.
     - By default, the root virtual machine folder is used.
  6. Role:
     - The role must include the required set of privileges.
  7. User:
     - The user must be assigned the role specified in the previous item.
- A tag from the category specified in the `regionTagCategory` parameter must be assigned to the created Datacenter (default: `k8s-region`). This tag defines the region.

List of required vSphere resources

- **User** with required set of privileges.
- **Network** with DHCP server and access to the Internet.
- **Datacenter** with a tag in `k8s-region` category.
- **Cluster** with a tag in `k8s-zone` category.
- **Datastore** with required tags.
- **Template** — prepared VM image.

List of required privileges

> **i** Read the Configuration via vSphere Client and Configuration via govc sections for details on how to create and assign a role to a user.

A detailed list of privileges required for Deckhouse Platform to work in vSphere:

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

—	— (assigned by default when creating a role)	`System.Anonymous`
		`System.Read`
		`System.View`
			Basic access to vSphere Inventory objects required for all Deckhouse vSphere integration components.

Cns	Searchable	`Cns.Searchable`	Search and mapping of Container Native Storage objects when the CSI driver works with Kubernetes volumes.

Datastore	Allocate space,
	Browse datastore,
	Low level file operations
		`Datastore.AllocateSpace`
		`Datastore.Browse`
		`Datastore.FileManagement`
			Disk provisioning when creating virtual machines and ordering `PersistentVolumes` in the cluster.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Folder	Create folder,
	Delete folder,
	Move folder,
	Rename folder
		`Folder.Create`
		`Folder.Delete`
		`Folder.Move`
		`Folder.Rename`
			Grouping a Deckhouse Platform cluster in a single `Folder` in vSphere Inventory.

Global	Global tag, System tag	`Global.GlobalTag` `Global.SystemTag`	Access to global and system tags used by Deckhouse Platform when working with vSphere objects.

vSphere Tagging	Assign or Unassign vSphere Tag,
	Assign or Unassign vSphere Tag on Object,
	Create vSphere Tag,
	Create vSphere Tag Category,
	Delete vSphere Tag,
	Delete vSphere Tag Category,
	Edit vSphere Tag,
	Edit vSphere Tag Category,
	Modify UsedBy Field for Category,
	Modify UsedBy Field for Tag
		`InventoryService.Tagging.AttachTag`
		`InventoryService.Tagging.ObjectAttachable`
		`InventoryService.Tagging.CreateTag`
		`InventoryService.Tagging.CreateCategory`
		`InventoryService.Tagging.DeleteTag`
		`InventoryService.Tagging.DeleteCategory`
		`InventoryService.Tagging.EditTag`
		`InventoryService.Tagging.EditCategory`
		`InventoryService.Tagging.ModifyUsedByForCategory`
		`InventoryService.Tagging.ModifyUsedByForTag`
			Deckhouse Platform uses tags to identify the `Datacenter`, `Cluster`, and `Datastore` objects available to it, as well as to identify the virtual machines under its control.

Network	Assign network	`Network.Assign`	Connecting networks and port groups to Deckhouse Platform cluster virtual machines.

Resource	Assign virtual machine to resource pool,
	Create resource pool,
	Modify resource pool,
	Remove resource pool,
	Rename resource pool
		`Resource.AssignVMToPool`
		`Resource.CreatePool`
		`Resource.DeletePool`
		`Resource.EditPool`
		`Resource.RenamePool`
			Placement of Deckhouse Platform cluster virtual machines into the target resource pool and management of this pool.

VM Storage Policies (*Profile-driven Storage Privileges* in vSphere 7)	View VM storage policies (*Profile-driven storage view* in vSphere 7)	`StorageProfile.View`	Viewing storage policies used when creating virtual machines and dynamically provisioning volumes in the cluster.

vApp	Add virtual machine,
	Assign resource pool,
	Create,
	Delete,
	Import,
	Power Off,
	Power On,
	View OVF Environment,
	vApp application configuration,
	vApp instance configuration,
	vApp resource configuration
		`VApp.ApplicationConfig`
		`VApp.AssignResourcePool`
		`VApp.AssignVM`
		`VApp.Create`
		`VApp.Delete`
		`VApp.ExtractOvfEnvironment`
		`VApp.Import`
		`VApp.InstanceConfig`
		`VApp.PowerOff`
		`VApp.PowerOn`
		`VApp.ResourceConfig`
			Managing operations related to deployment and configuration of vApp and OVF templates used when creating virtual machines.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Change Configuration	Add existing disk,
	Add new disk,
	Add or remove device,
	Advanced configuration,
	Set annotation,
	Change CPU count,
	Toggle disk change tracking,
	Extend virtual disk,
	Acquire disk lease,
	Modify device settings,
	Configure managedBy,
	Change Memory,
	Query unowned files,
	Configure Raw device,
	Reload from path,
	Remove disk,
	Rename,
	Reset guest information,
	Change resource,
	Change Settings,
	Change Swapfile placement,
	Upgrade virtual machine compatibility
		`VirtualMachine.Config.AddExistingDisk`
		`VirtualMachine.Config.AddNewDisk`
		`VirtualMachine.Config.AddRemoveDevice`
		`VirtualMachine.Config.AdvancedConfig`
		`VirtualMachine.Config.Annotation`
		`VirtualMachine.Config.CPUCount`
		`VirtualMachine.Config.ChangeTracking`
		`VirtualMachine.Config.DiskExtend`
		`VirtualMachine.Config.DiskLease`
		`VirtualMachine.Config.EditDevice`
		`VirtualMachine.Config.ManagedBy`
		`VirtualMachine.Config.Memory`
		`VirtualMachine.Config.QueryUnownedFiles`
		`VirtualMachine.Config.RawDevice`
		`VirtualMachine.Config.ReloadFromPath`
		`VirtualMachine.Config.RemoveDisk`
		`VirtualMachine.Config.Rename`
		`VirtualMachine.Config.ResetGuestInfo`
		`VirtualMachine.Config.Resource`
		`VirtualMachine.Config.Settings`
		`VirtualMachine.Config.SwapPlacement`
		`VirtualMachine.Config.UpgradeVirtualHardware`
			Managing the lifecycle of Deckhouse Platform cluster virtual machines.

Virtual Machine > Edit Inventory	Create new,
	Create from existing,
	Remove,
	Move
		`VirtualMachine.Inventory.Create`
		`VirtualMachine.Inventory.CreateFromExisting`
		`VirtualMachine.Inventory.Delete`
		`VirtualMachine.Inventory.Move`
			Creating, deleting, and moving Deckhouse Platform cluster virtual machines in vSphere Inventory.

Virtual Machine > Guest Operations	Guest Operation Queries	`VirtualMachine.GuestOperations.Query`	Retrieving information from the guest operating system of virtual machines.

Virtual Machine > Interaction	Answer question,
	Device connection,
	Guest operating system management by VIX API,
	Power Off,
	Power On,
	Reset,
	Configure CD media,
	Install VMware Tools
		`VirtualMachine.Interact.AnswerQuestion`
		`VirtualMachine.Interact.DeviceConnection`
		`VirtualMachine.Interact.GuestControl`
		`VirtualMachine.Interact.PowerOff`
		`VirtualMachine.Interact.PowerOn`
		`VirtualMachine.Interact.Reset`
		`VirtualMachine.Interact.SetCDMedia`
		`VirtualMachine.Interact.ToolsInstall`
			Managing virtual machine power state, device connections, and interaction with the guest operating system.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Provisioning	Clone virtual machine,
	Customize guest,
	Deploy template,
	Allow virtual machine download,
	Allow virtual machine files upload,
	Read customization specifications
		`VirtualMachine.Provisioning.Clone`
		`VirtualMachine.Provisioning.Customize`
		`VirtualMachine.Provisioning.DeployTemplate`
		`VirtualMachine.Provisioning.GetVmFiles`
		`VirtualMachine.Provisioning.PutVmFiles`
		`VirtualMachine.Provisioning.ReadCustSpecs`
			Cloning virtual machine templates, customizing them, and deploying them when creating Deckhouse Platform cluster nodes.

Virtual Machine > Snapshot Management	Create snapshot,
	Remove Snapshot,
	Rename Snapshot
		`VirtualMachine.State.CreateSnapshot`
		`VirtualMachine.State.RemoveSnapshot`
		`VirtualMachine.State.RenameSnapshot`
			Managing snapshots of virtual machines and volumes in scenarios where this functionality is used by platform components.

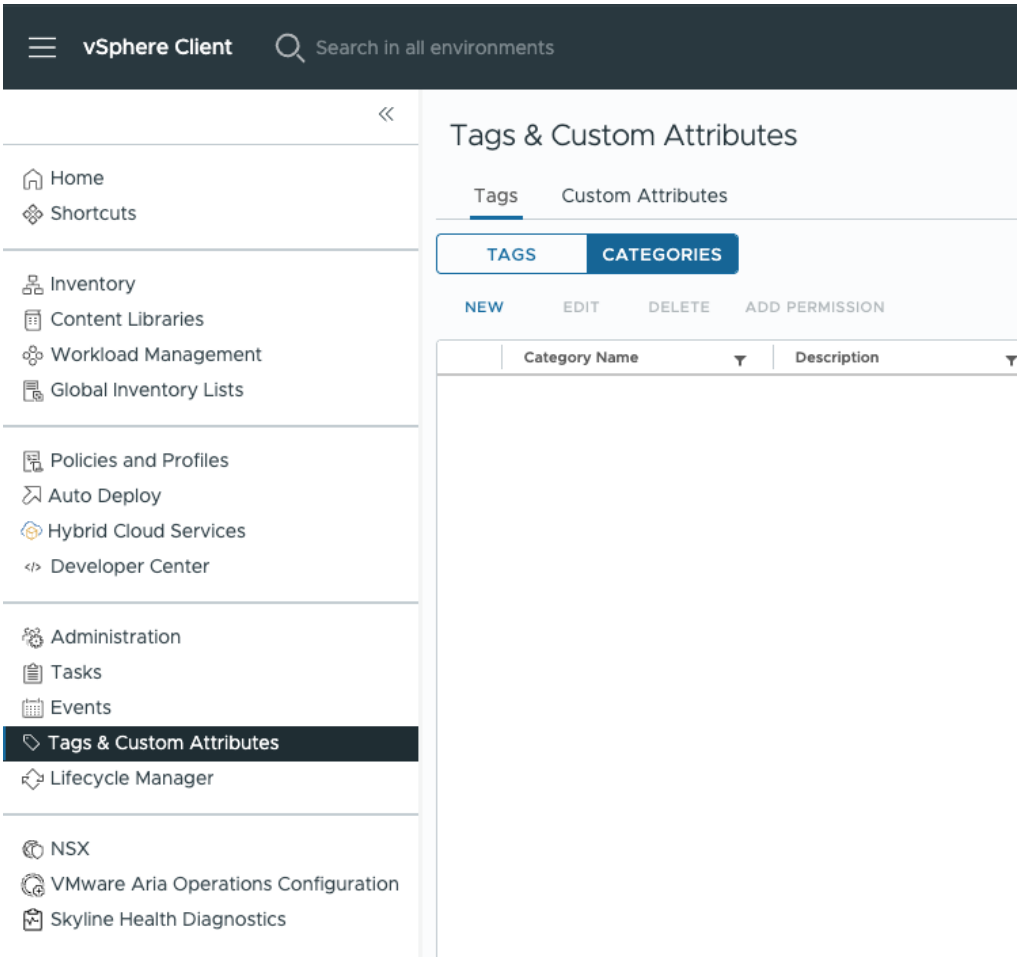
vSphere configuration

Configuration in vSphere Client

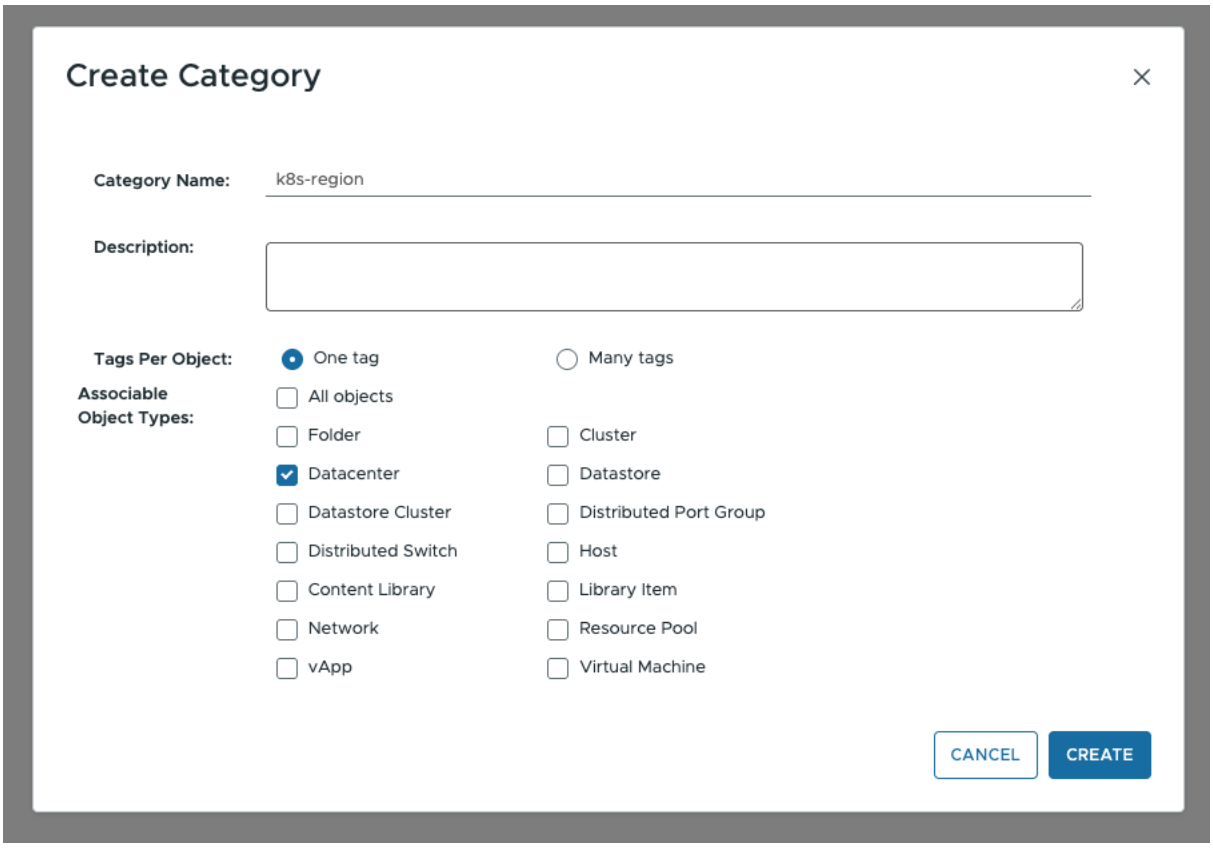
Creating tags and tag categories in vSphere Client

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

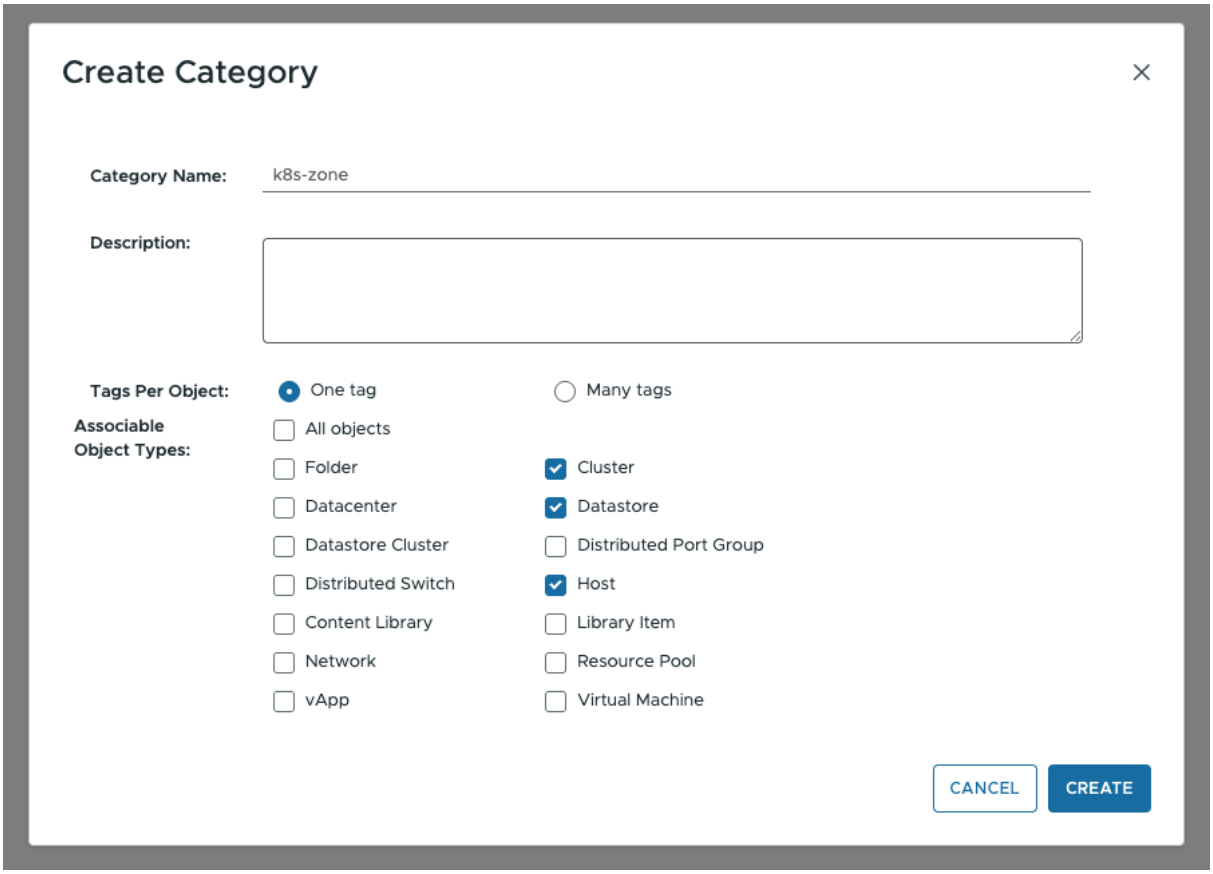
1. Open vSphere Client and go to “Menu” → “Tags & Custom Attributes” → “Tags”.



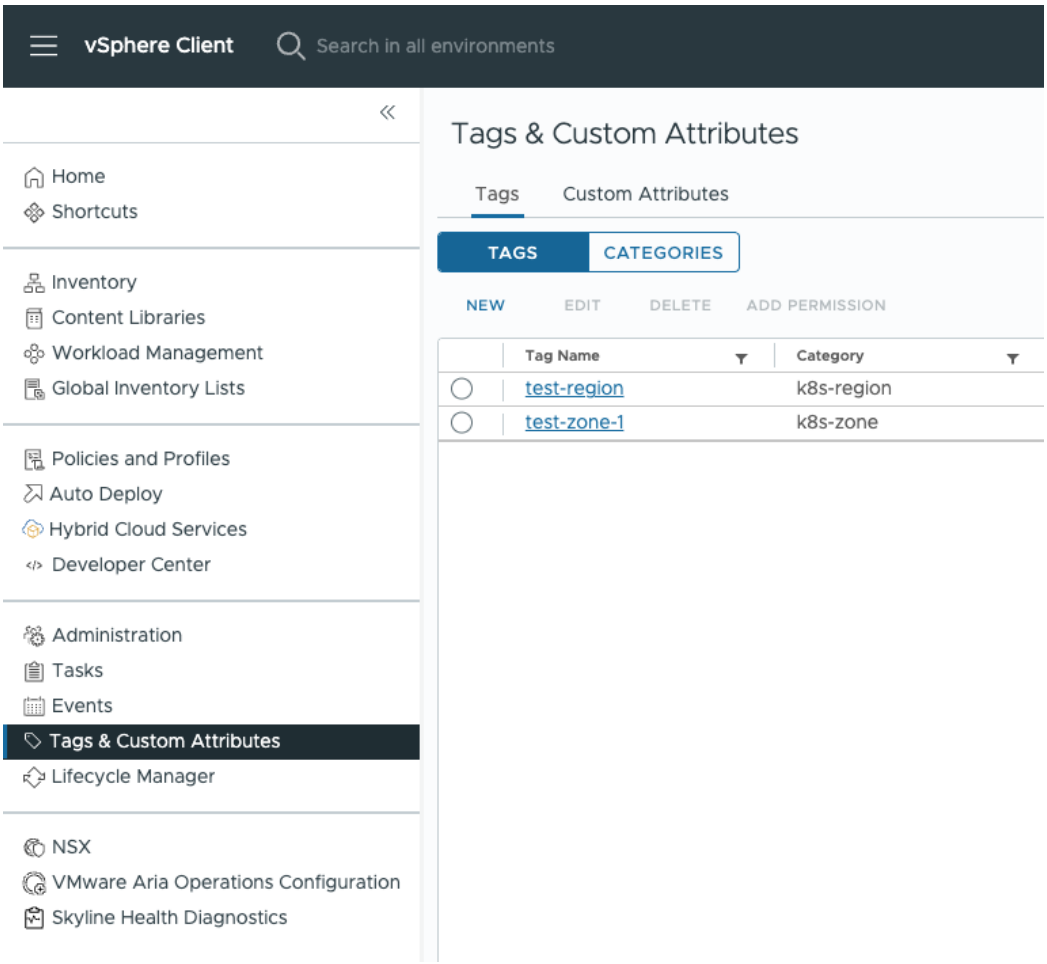
2. Open the “Categories” tab and click “NEW”. Create a category for regions (for example, `k8s-region`): set “Tags Per Object” to “One tag” and specify the applicable object types, including Datacenter.



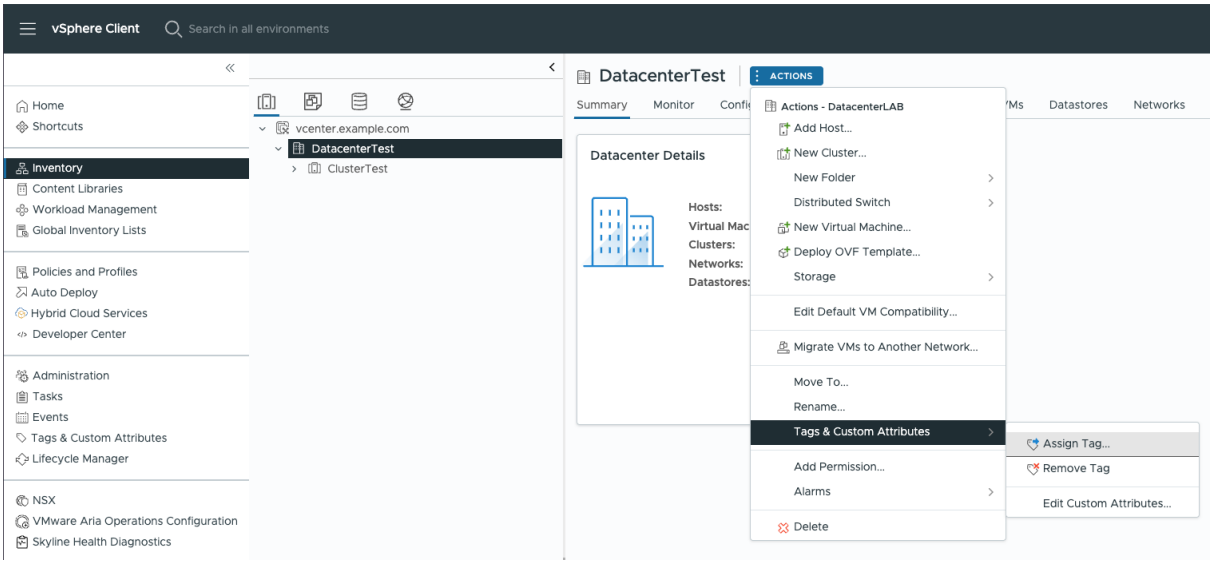
3. Create a second category for zones (for example, `k8s-zone`) with the object types Host, Cluster, and Datastore.

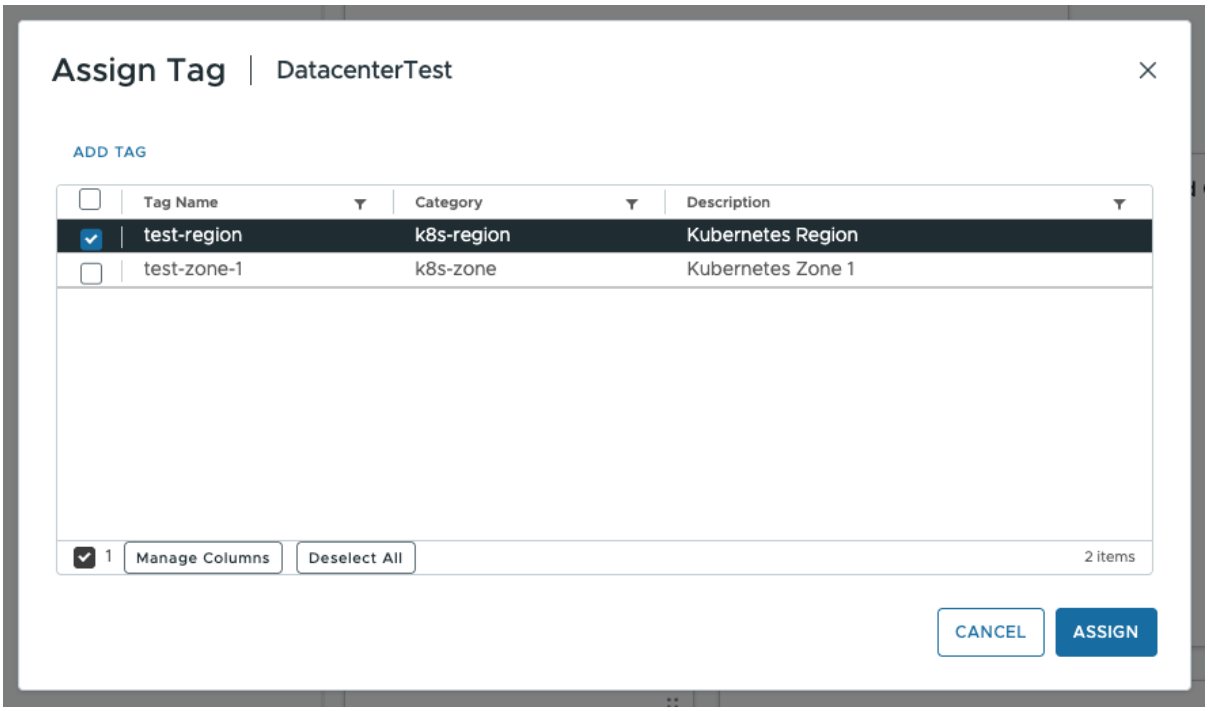


4. Go to the “Tags” tab and create at least one tag in the region category and one tag in the zone category (for example, `test-region`, `test-zone-1`).



5. In the “Inventory” tab, select the target Datacenter, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag” and assign the region tag. Repeat this step for each Cluster that will host nodes, assigning the appropriate zone tags.

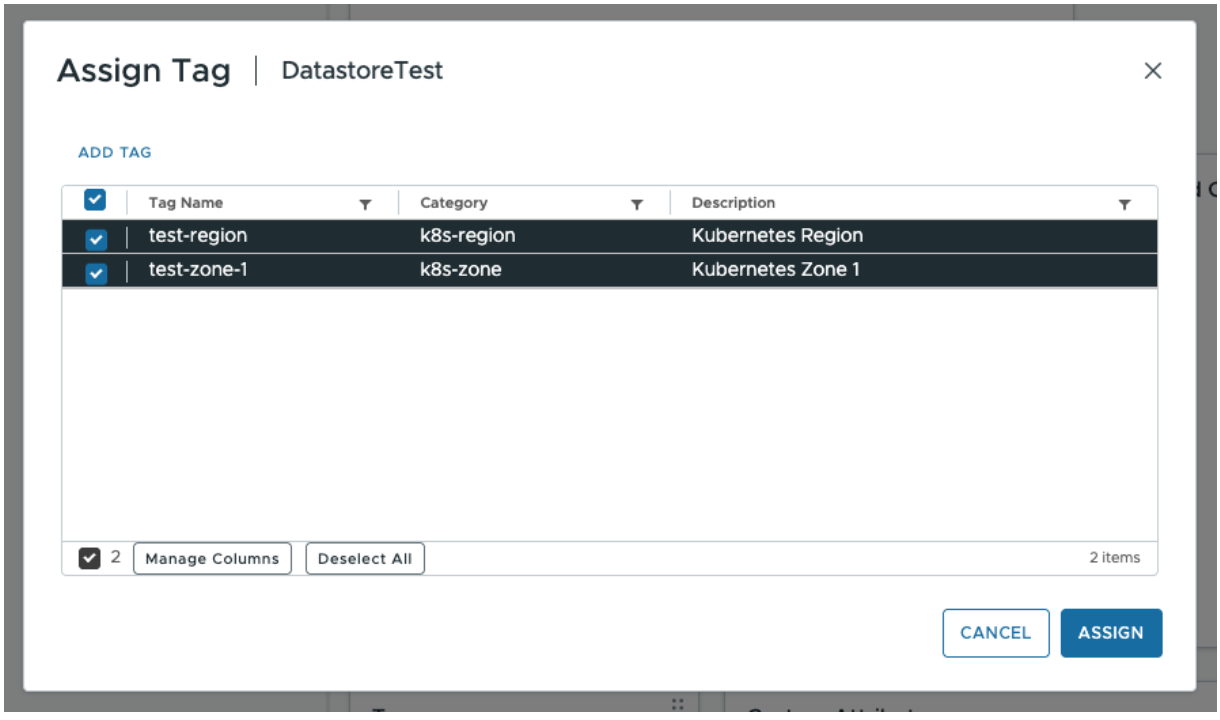




Configuring Datastore in vSphere Client

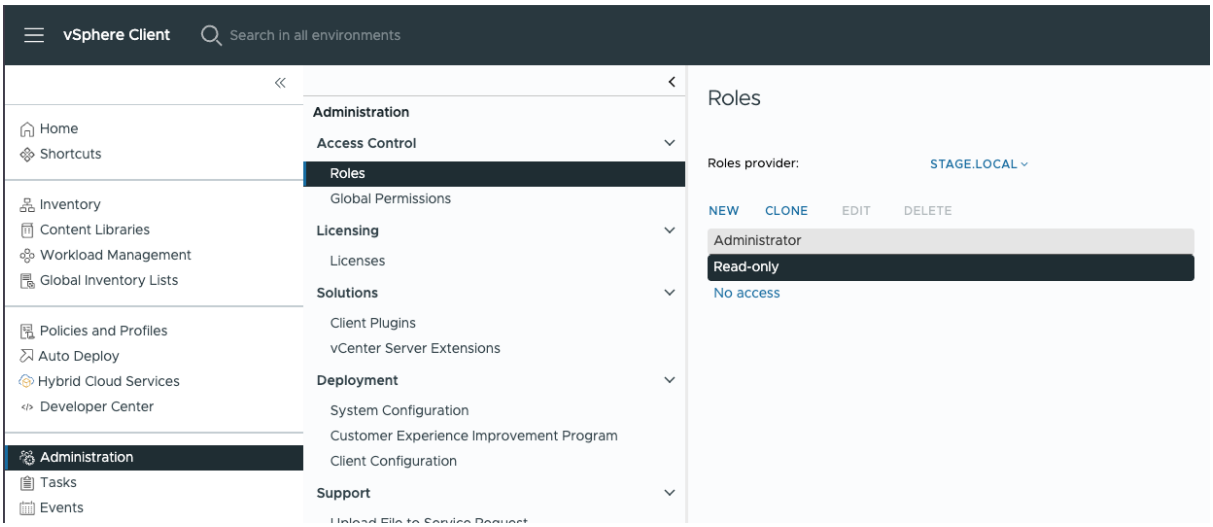
> ⚠ For dynamic provisioning of PersistentVolume, the Datastore must be available on **every** ESXi host in the zone (shared datastore).

In the “Inventory” tab, select the Datastore, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag”. Assign the Datastore the same region tag as the corresponding Datacenter, and the same zone tag as the corresponding Cluster.

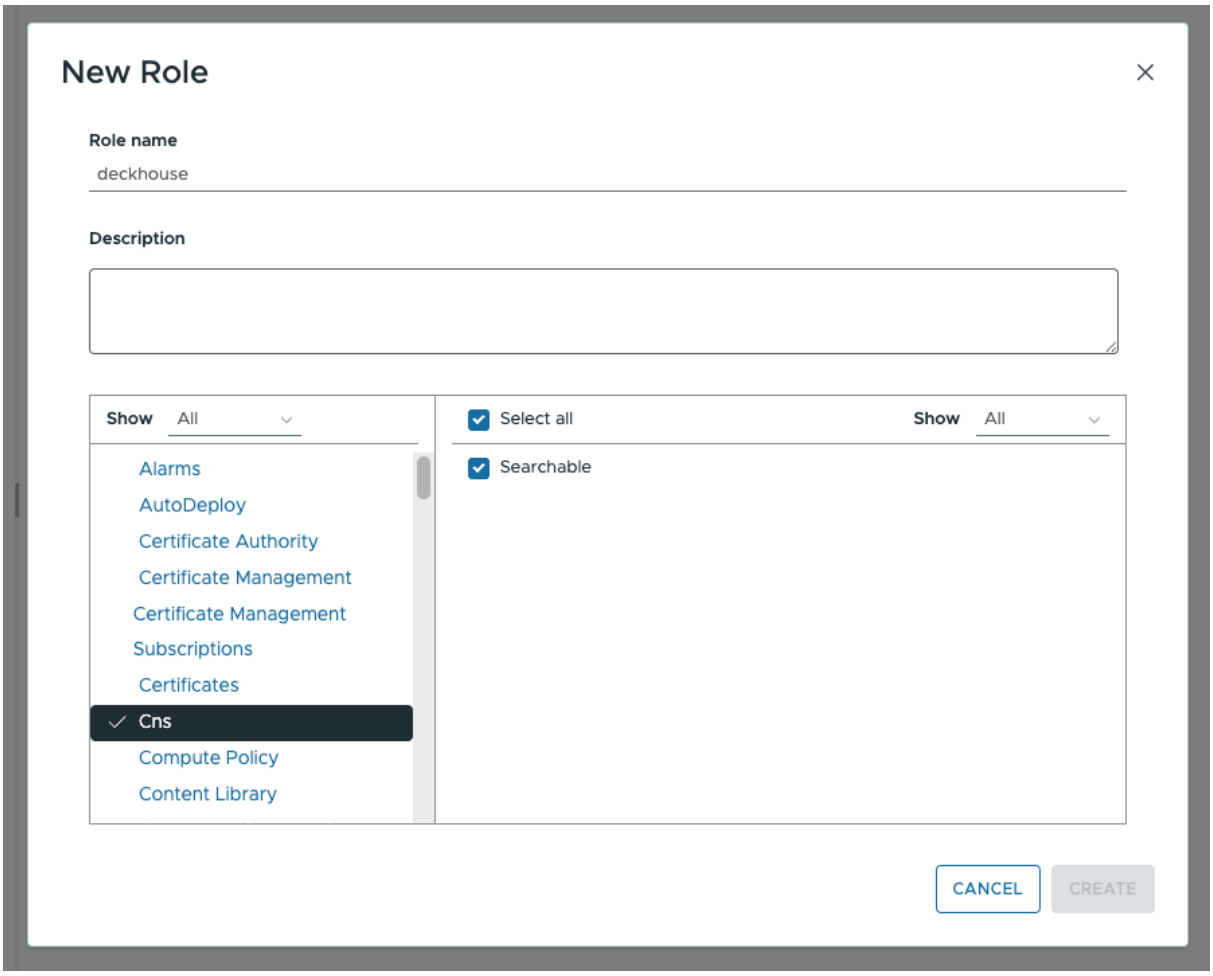


Creating and assigning a role in vSphere Client

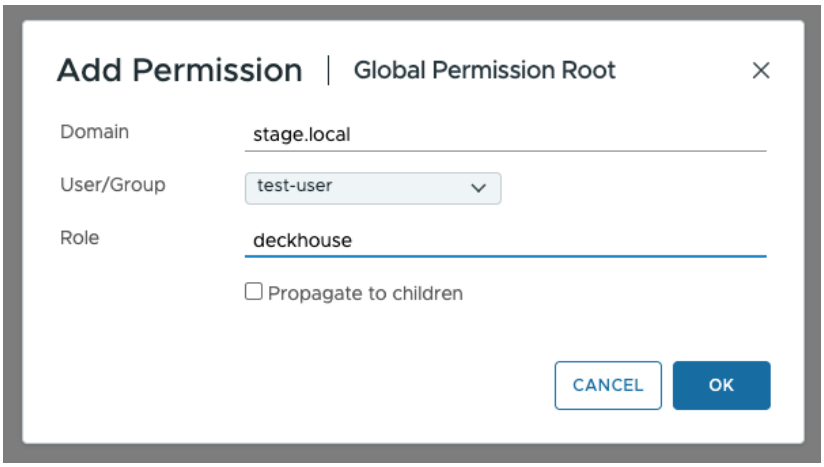
1. Go to “Menu” → “Administration” → “Access Control” → “Roles”.



2. Click “NEW”, enter a role name (for example, `deckhouse`), and add the privileges from the list.



3. Assign the role to the Deckhouse service account: go to “Menu” → “Administration” → “Access Control” → “Global Permissions”, click “ADD”, and select the user and the `deckhouse` role.



Configuration via govc

Installing govc

To continue configuring vSphere, install the govc CLI utility.

After installation, set the environment variables required to connect to `vCenter`.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Creating tags and tag categories with govc

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

Create tag categories with the following commands:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Create tags in each category. If you intend to use multiple “zones” (Cluster), create a tag for each one of them:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Attach the “region” tag to Datacenter:

```
govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>
```

Attach “zone” tags to the Cluster objects:

```
govc tags.attach -C k8s-zone test-zone-1 /<DATACENTER_NAME>/host/<CLUSTER_NAME_1>
govc tags.attach -C k8s-zone test-zone-2 /<DATACENTER_NAME>/host/<CLUSTER_NAME_2>
```

Datastore configuration with govc

⚠ For dynamic PersistentVolume provisioning, a Datastore must be available on **each** ESXi host (shared datastore).

Assign the “region” and “zone” tags to the Datastore objects to automatically create a StorageClass in the Kubernetes cluster:

```
govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_1>
govc tags.attach -C k8s-zone test-zone-1 /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_1>

govc tags.attach -C k8s-region test-region /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_2>
govc tags.attach -C k8s-zone test-zone-2 /<DATACENTER_NAME>/datastore/<DATASTORE_NAME_2>
```

Creating and assigning a role with govc

> **i** We've intentionally skipped User creation since there are many ways to authenticate a user in the vSphere.
>
> The role described below includes the privileges from the list of required privileges section. If you need a more granular Role, please contact your Deckhouse support.

Create a role with the required privileges:

```
govc role.create deckhouse \
    Cns.Searchable \
    Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
    Folder.Create Folder.Delete Folder.Move Folder.Rename \
    Global.GlobalTag Global.SystemTag \
    InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
    InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
    InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
    InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
    InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
    Network.Assign \
    Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
    StorageProfile.View \
    System.Anonymous System.Read System.View \
    VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
    VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
    VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
    VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
    VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
    VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
    VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
    VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
    VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
    VirtualMachine.Config.UpgradeVirtualHardware \
    VirtualMachine.GuestOperations.Query \
    VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
    VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
    VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
    VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
    VirtualMachine.Inventory.Move \
    VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
    VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs
\
    VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Assign the role to a user on the vCenter object.

> **⚠** Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

> **i** For more detailed permission configuration, refer to the official documentation.

VM image requirements

To create a VM template (`Template`), it is recommended to use a ready-made cloud image/OVA file provided by the OS vendor:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (section *Generic Cloud / OpenStack*)

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Preparing the virtual machine image

⚠ Disable VMware Guest OS Customization (and any vApp/OS customization mechanisms, if applicable in your setup) for the template and the cluster virtual machines. DP performs the initial node configuration via `cloud-init` (VMware GuestInfo datasource). Enabled customization can conflict with `cloud-init` and lead to incorrect node initialization.

1. Install the required packages:

   If you use `cloud-init` version lower than 21.3 (VMware GuestInfo support is required):

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
   ```

   If you use `cloud-init` version 21.3 or higher:

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init
   ```

2. Verify that the `disable_vmware_customization: false` parameter is set in `/etc/cloud/cloud.cfg`.
3. Make sure the `default_user` parameter is specified in `/etc/cloud/cloud.cfg`. It is required to add an SSH key when the VM starts.
4. Add the VMware GuestInfo datasource — create `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

   ```
   datasource:
     VMware:
       vmware_cust_file_max_wait: 10
   ```

5. Before creating the VM template, reset the `cloud-init` identifiers and state using the following commands:

   ```
   truncate -s 0 /etc/machine-id &&
   rm /var/lib/dbus/machine-id &&
   ln -s /etc/machine-id /var/lib/dbus/machine-id
   ```

6. Clear `cloud-init` event logs:

   ```
   cloud-init clean --logs --seed
   ```

⚠ After the virtual machine starts, the following services related to the packages installed during `cloud-init` preparation must be running:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

To ensure that the services are enabled, use the command:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Example output for enabled services:

```
enabled
enabled
enabled
```

i DP creates VM disks of type `eagerZeroedThick`, but the type of disks of created VMs may be changed without notification according to the `VM Storage Policy` settings in vSphere.
For more details, see the documentation.

i DP uses the `ens192` interface as the default interface for VMs in vSphere. Therefore, when using static IP addresses in `mainNetwork`, you must create an interface named `ens192` in the OS image as the default interface.

Replacing the OS image

Deckhouse Platform creates the nodes from a virtual machine template that is specified in the `template` parameter of the VsphereInstanceClass resource for `CloudEphemeral` nodes and in the `instanceClass` section of the VsphereClusterConfiguration resource for `CloudPermanent` nodes and master nodes.

To move the nodes to a new OS image, prepare a new template based on it and follow the steps from “How do I replace the OS image in a cloud cluster?”.

Infrastructure

Networking

A VLAN with DHCP and Internet access is required for the running cluster:

- If the VLAN is public (public addresses), then you have to create a second network to deploy cluster nodes (DHCP is not needed in this network).
- If the VLAN is private (private addresses), then this network can be used for cluster nodes.

Inbound traffic

- You can use an internal load balancer (if present) and direct traffic directly to the front nodes of the cluster.
- If there is no load balancer, you can use MetalLB in BGP mode to organize fault-tolerant load balancers (recommended). In this case, front nodes of the cluster will have two interfaces. For this, you will need:
  - A dedicated VLAN for traffic exchange between BGP routers and MetalLB. This VLAN must have DHCP and Internet access.
  - IP addresses of BGP routers.
  - ASN — the AS number on the BGP router.
  - ASN — the AS number in the cluster.
  - A range to announce addresses from.

Using the datastore

Various types of storage can be used in the cluster; for the minimum configuration, you will need:

- Datastore for provisioning PersistentVolumes to the Kubernetes cluster.
- Datastore for provisioning root disks for the VMs (it can be the same Datastore as for PersistentVolume).

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

VsphereInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of vSphere VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines’ secondary NICs will connect to. Relative to the datacenter.

    Examples:

    ```
    additionalNetworks: DEVOPS_32
    ```

```
additionalNetworks: DEVOPS_50
```

- **spec.datastore**
  string

  Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

  Both a plain Datastore name (`lun-1201`) and a path with parent folders (`DS_PARENT_FOLDER/lun-1201`) are supported.

  By default, a datastore from the `master` InstanceClass is used.

  Example:

  ```
  datastore: lun-1201
  ```

- **spec.disableTimesync**
  boolean

  Disable time synchronization in Guest VM.

  > **Note:** this won't disable any NTP daemons running on the guest OS, it only disables time synchronization from ESXi.

  Default: `true`

- **spec.mainNetwork**
  string

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

  If no value is specified, the value specified in `VsphereClusterConfiguration` will be used.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **spec.memory**
  integer

  Required value

  Memory in MiB to allocate to vSphere VirtualMachines.

  Example:

  ```
  memory: 8192
  ```

- **spec.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **spec.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  By default, the value from `VsphereCloudDiscoveryData` is used.

  Example:

  ```
  resourcePool: rp-2012
  ```

- **spec.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Default: `20`

  Example:

  ```
  rootDiskSize: 20
  ```

- **spec.runtimeOptions**
  object

  Additional VM’s parameters.

  - **spec.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **spec.runtimeOptions.cpuReservation**
    integer

    CPU reservation in MHz.

  - **spec.runtimeOptions.cpuShares**
    integer

    The relative amount of CPU Shares for VMs to be created.

  - **spec.runtimeOptions.memoryLimit**
    integer

    Memory limit in MB.

  - **spec.runtimeOptions.memoryReservation**
    integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Default: `80`

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.memoryShares**
    integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.nestedHardwareVirtualization**
    boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

- **spec.template**
  string

  Path to the template to be cloned. Relative to the datacenter.

  By default, a template from the `master` InstanceClass is used.

  Example:

  ```
  template: dev/golden_image
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of vSphere VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines’ secondary NICs will connect to. Relative to the datacenter.

    Examples:

    ```
    additionalNetworks: DEVOPS_32
    ```

    ```
    additionalNetworks: DEVOPS_50
    ```

  - **spec.datastore**
    string

    Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

    Both a plain Datastore name (`lun-1201`) and a path with parent folders (`DS_PARENT_FOLDER/lun-1201`) are supported.

By default, a datastore from the `master` InstanceClass is used.

Example:

```
datastore: lun-1201
```

- **spec.disableTimesync**
  boolean

  Disable time synchronization in Guest VM.

  > **i** **Note:** this won't disable any NTP daemons running on the guest OS, it only disables time synchronization from ESXi.

  Default: `true`

- **spec.mainNetwork**
  string

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

  If no value is specified, the value specified in `VsphereClusterConfiguration` will be used.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **spec.memory**
  integer

  Required value

  Memory in MiB to allocate to vSphere VirtualMachines.

  Example:

  ```
  memory: 8192
  ```

- **spec.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **spec.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  By default, the value from `VsphereCloudDiscoveryData` is used.

  Example:

  ```
  resourcePool: rp-2012
  ```

- **spec.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Default: `20`

  Example:

  ```
  rootDiskSize: 20
  ```

- **spec.runtimeOptions**
  object

  Additional VM's parameters.

  - **spec.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **spec.runtimeOptions.cpuReservation**

integer

CPU reservation in MHz.

- **spec.runtimeOptions.cpuShares**
  integer

  The relative amount of CPU Shares for VMs to be created.

- **spec.runtimeOptions.memoryLimit**
  integer

  Memory limit in MB.

- **spec.runtimeOptions.memoryReservation**
  integer

  VM memory reservation in percent (relative to `.spec.memory`).

  Default: `80`

  Allowed values: `0 <= X <= 100`

- **spec.runtimeOptions.memoryShares**
  integer

  The relative amount of Memory Shares for VMs to be created.

  Allowed values: `0 <= X <= 100`

- **spec.runtimeOptions.nestedHardwareVirtualization**
  boolean

  Whether to enable or disable nested Hardware Assisted Virtualization.

- **spec.template**
  string

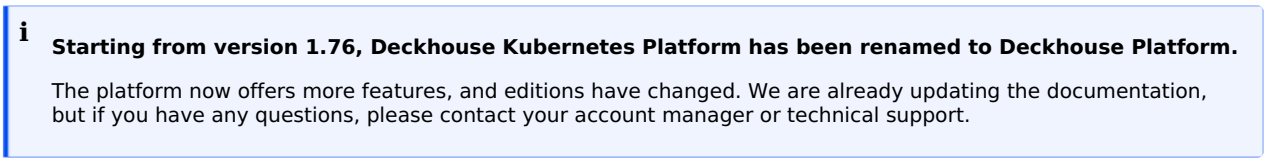
  Path to the template to be cloned. Relative to the datacenter.

  By default, a template from the `master` InstanceClass is used.

  Example:

  ```
  template: dev/golden_image
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings



Examples



Below is an example configuration for a VMware vSphere cloud provider.

An example of the configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: vc-3.internal
    username: user
    password: password
    vmFolderPath: dev/test
    insecure: true
    region: moscow-x001
    sshKeys:
    - "ssh-rsa AAAAB3N....6xHJwwj"
    externalNetworkNames:
    - KUBE-3
    - devops-internal
    internalNetworkNames:
    - KUBE-3
    - devops-internal
    nsxt:
      defaultIpPoolName: "External IP Pool"
      tier1GatewayPath: flant_tier1
      user: guestuser1
      password: pass
      host: 1.2.3.4
      insecureFlag: true
      size: SMALL
```

An example of the `VsphereInstanceClass` custom resource

```
apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: test
spec:
  numCPUs: 2
  memory: 2048
  rootDiskSize: 20
  template: dev/golden_image
  mainNetwork: k8s-msk-178
  datastore: <DS_PARENT_FOLDER>/lun-1201 # <DS_PARENT_FOLDER> — parent Datastore folder.
```

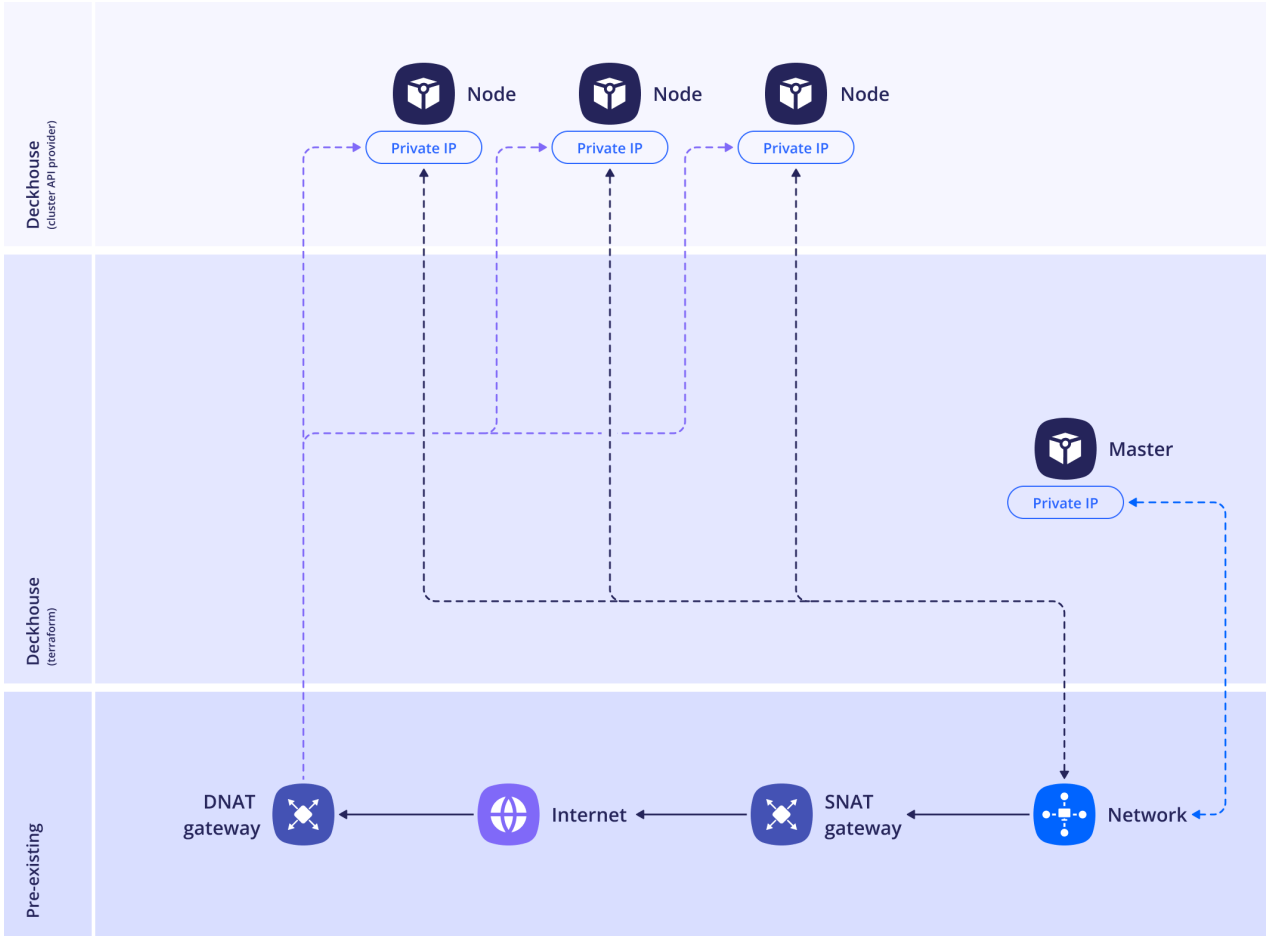
Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider - VMware vSphere: Layouts

The module lifecycle stage: General Availability

Standard



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - ru-central1-a
  - ru-central1-b
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
nodeGroups:
- name: khm
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- ru-central1-a
- ru-central1-b
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — Yandex Cloud

The module lifecycle stage: General Availability

The `cloud-provider-yandex` module integrates Deckhouse Platform with Yandex Cloud. It allows the `node-manager` module to use Yandex Cloud resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-yandex` module:

- Managing Yandex Cloud resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the Yandex Cloud side;
  - creates Network Load Balancers and target groups for Services of the LoadBalancer type;
  - updates instance and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in Yandex Cloud.
- Provisioning disks via the Yandex CSI driver (`yandex.csi.flant.com`) and creating StorageClasses for Yandex Cloud disk types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-yandex`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM) or Cluster API (CAPI). Virtual machine parameters are set in the YandexInstanceClass resource.
- Registering with `node-manager` so that YandexInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. Starting with DP 1.76, `cni-cilium` is used by default.

⚠ If the cluster was installed with the YandexClusterConfiguration schema, it must be migrated to the ModuleConfig-based configuration. Until the migration is complete, the `D8CloudProviderYandexCloudMigrationPending` alert fires and a Deckhouse Platform update can be blocked.

The migration procedure is described in “How to migrate a cloud provider to ModuleConfig-based configuration”.

⚠ The module is migrating CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI). Existing NodeGroups continue to use MCM, while new ones are created with CAPI by default. For migrating existing groups, see “How to migrate node groups to Cluster API (CAPI)”.

Third-party components

List of third-party software used in the `cloud-provider-yandex` module:

- Yandex Cloud Terraform provider 0.174.0

  License: Mozilla Public License 2.0

  Terraform provider for Yandex Cloud.

- Yandex Cloud provider v0.33.7

  License: Apache License 2.0

  Cloud Controller Manager for Yandex Cloud.

- Yandex Cloud CSI driver v0.16.1

  License: Apache License 2.0

  CSI driver for Yandex Compute Disks.

- Yandex Cloud Cluster API provider v0.2.0

  License: Apache License 2.0

  Infrastructure provider for Kubernetes Cluster API in Yandex Cloud.

ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

ℹ The module lifecycle stage: General Availability

Deckhouse Platform reads the credentials for accessing the Yandex Cloud API from the `d8-credentials` Secret, whose format is described in “Credentials Secret”.

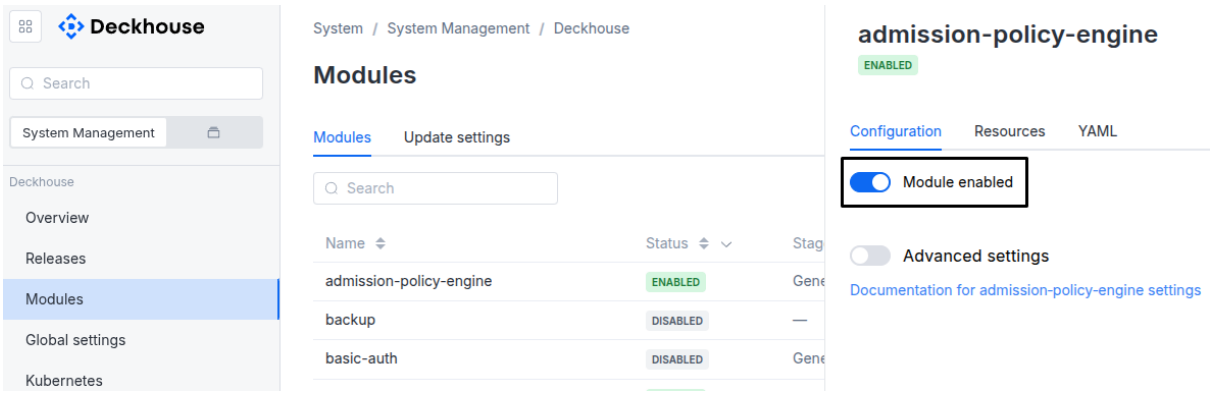
⚠ When the module settings change, the existing Machine objects are not recreated, and only new objects are created with the new parameters. Nodes are recreated when the parameters of NodeGroup and YandexInstanceClass change. For recreating nodes manually, see “How do I redeploy ephemeral machines in the cloud with a new configuration?”.

The module has 4 alerts.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-yandex` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-yandex
  ```

- **Using ModuleConfig `cloud-provider-yandex`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-yandex` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-yandex`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-yandex
  spec:
    enabled: true
  ```

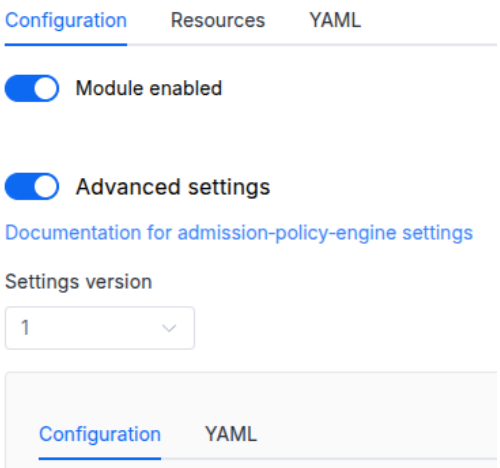
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-yandex` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-yandex` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-yandex`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-yandex` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-yandex
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the

`spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-yandex` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-yandex`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  The settings are grouped under the `provider`, `nodes`, `storage`, and `ccm` parameters. The conversion does the following:

  - The `cloudID`, `folderID`, `layout`, `nodeNetworkCIDR`, and `sshPublicKey` parameters have to be filled in manually.
  - The `storageClass.exclude` parameter is renamed to `excludedStorageClasses`, and `storageClass.provision` to `provisionedStorageClasses`.
  - The `storageClass.default` parameter is removed. Set the default StorageClass with the global `defaultClusterStorageClass` parameter.
  - The `additionalExternalNetworkIDs` parameter is moved to `ccm.parameters.additionalExternalNetworkIDs`.

Module configuration

Every provider setting except the credentials is defined in the module’s ModuleConfig resource. Deckhouse Platform applies the settings both when creating a cluster and in a running cluster.

The settings are grouped under four top-level parameters:

- `provider` sets the Yandex Cloud cloud and folder IDs, the parameter is required;
- `nodes` sets the layout, the network parameters, and the parameters of the nodes being created, the parameter is required;
- `storage` sets the StorageClass the module creates;
- `ccm` sets how the cloud controller manager works.

The nodes, storage, and cloud controller manager subsystems are turned off with the nested `disabled` parameter.

Example module configuration with provider, node, storage, and cloud controller manager settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 10.241.32.0/20
        sshPublicKey: <SSH_PUBLIC_KEY>
    storage:
      parameters:
        excludedStorageClasses:
          - network-hdd
    ccm:
      parameters:
        additionalExternalNetworkIDs:
          - enp6t4snovl2ko4p15em
```

Where:

- `CLOUD ID` and `FOLDER_ID` are the Yandex Cloud cloud and folder IDs;
- `SSH_PUBLIC_KEY` is the public SSH key for accessing the nodes.

The virtual machine parameters are set in the YandexInstanceClass resource that a NodeGroup refers to.

Storage

The module automatically creates a StorageClass for every Yandex Cloud disk type:

Type	StorageClass name	Note
network-hdd	network-hdd	
network-ssd	network-ssd	
network-ssd-nonreplicated	network-ssd-nonreplicated	Disk size must be a multiple of 93 GB.
network-ssd-io-m3	network-ssd-io-m3	Disk size must be a multiple of 93 GB.

You can filter out the unnecessary StorageClasses via the `excludedStorageClasses` parameter.

The `provisionedStorageClasses` parameter lets you create additional StorageClasses or override the parameters of StorageClasses created by the module by default.

Use the `blockSize` parameter to set the block size for provisioned disks. The block size determines the maximum disk size. It is `8Ti` for `4Ki` and doubles with each next block size, up to `256Ti` for `128Ki`.

Example configuration of the `cloud-provider-yandex` module with an additional StorageClass configured with a block size of `64Ki`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 192.168.12.0/24
        sshPublicKey: <SSH_PUBLIC_KEY>
    storage:
      parameters:
        provisionedStorageClasses:
          - name: network-ssd-64k
            type: network-ssd
            blockSize: 64Ki
```

⚠ After a disk is created, its block size cannot be changed. Changing the `blockSize` parameter recreates the StorageClass, but does not change the block size of previously provisioned volumes.

LoadBalancer

The module watches Service objects of the `LoadBalancer` type and creates a network load balancer and a target group for them in Yandex Cloud. The annotations that configure the balancer are listed in “Service annotations”.

For more information, see the Kubernetes Cloud Controller Manager for Yandex Cloud documentation.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.ccm**
    object

    Cloud Controller Manager (CCM) subsystem settings.

    CCM creates Network Load Balancers and target groups for Service objects of the `LoadBalancer` type, maintains routes to pod subnets in the route table, and keeps node metadata up to date.

    - **settings.ccm.disabled**
      boolean

      Disables the Cloud Controller Manager.

      When set to `true`, the cluster does not manage load balancers and routes in Yandex Cloud.

      Default: `false`

    - **settings.ccm.parameters**
      object

      Required value

      Parameters of the Cloud Controller Manager subsystem.

      - **settings.ccm.parameters.additionalExternalNetworkIDs**
        array of strings

        Additional external network IDs to be recognized as external networks by the CCM.

      - **settings.ccm.parameters.additionalInternalNetworkIDs**
        array of strings

        Additional internal network IDs to be recognized as internal networks by the CCM.

  - **settings.nodes**
    object

    Required value

    Nodes subsystem settings.

    The subsystem describes the infrastructure the cluster nodes are created in, including the layout, the networks,

the zones, and the SSH key.

- **settings.nodes.disabled**
  boolean

  Disables the nodes subsystem.

  When set to `true`, the cluster creates no nodes in Yandex Cloud, and the cloud controller manager does not start either.

  Default: `false`

- **settings.nodes.parameters**
  object

  Required value

  Parameters of the nodes subsystem.

  - **settings.nodes.parameters.dhcpOptions**
    object

    A list of DHCP parameters to use for all subnets.

    Setting the parameter can disrupt name resolution on the nodes. The symptoms and the ways to address them are described in DHCPOptions-related problems and ways to address them.

    - **settings.nodes.parameters.dhcpOptions.domainName**
      string

      Domain appended to an incomplete host name when the node resolves it.

    - **settings.nodes.parameters.dhcpOptions.domainNameServers**
      array of strings

      A list of recursive DNS addresses.

  - **settings.nodes.parameters.existingNetworkID**
    string

    The ID of the existing VPC Network.

  - **settings.nodes.parameters.existingRouteTableID**
    string

    The ID of the existing route table associated with the subnets.

    Normally the value is produced when DP creates the infrastructure. Set the parameter explicitly in a cluster where DP does not create the infrastructure, for example in a cluster with static master nodes that adds ephemeral nodes in Yandex Cloud.

  - **settings.nodes.parameters.existingZoneToSubnetIDMap**
    object

    One or more pre-existing subnets mapped to respective zone.

    Warning. With the `cni-simple-bridge` module, DP creates a route table that has to be associated with the listed subnets manually. A subnet accepts only one route table, so two clusters using `cni-simple-bridge` cannot share subnets. Starting with DP 1.76, new clusters in Yandex Cloud use the `cni-cilium` module in VXLAN mode, where pod traffic routing does not depend on Yandex Cloud route tables.

  - **settings.nodes.parameters.externalIPAddresses**
    object

    External IP addresses per node group name.

    The key is the node group name (e.g., `system`, `master`, `worker`), and the value is the list of external IP addresses for nodes in that group, listed in the order of the zones where nodes will be created.

    The following values can be specified in the list:

    - IP address from an additional external network for the corresponding zone, listed in the `externalSubnetIDs` parameter;
    - reserved public IP address, if the `externalSubnetIDs` parameter is not set;
    - `Auto`, to order a public IP address in the corresponding zone.

  - **settings.nodes.parameters.externalSubnetIDs**
    object

    IDs of additional external networks per node group name.

    The key is the node group name (e.g., `system`, `master`, `worker`), and the value is the list of external subnet IDs for nodes in that group, listed in the order of the zones where nodes will be created.

  - **settings.nodes.parameters.labels**
    object

    Labels to attach to resources created in the Yandex Cloud.

    Note that you have to re-create all the machines to add new labels if labels were modified in the

running cluster.

- **settings.nodes.parameters.layout**
  string

  Required value

  The way resources are located in the cloud.

  The available layouts are described in “Layouts”.

  Allowed values: `Standard`, `WithoutNAT`, `WithNATInstance`

- **settings.nodes.parameters.nodeNetworkCIDR**
  string

  Required value

  IP address range for node subnets.

  Deckhouse Platform divides the range into four equal parts:

  - `ru-central1-a` uses the first part.
  - `ru-central1-b` uses the second part.
  - `ru-central1-e` uses the third part.
  - `ru-central1-d` uses the fourth part.

  Instructions for using the `ru-central1-f` zone are provided in “Using the ru-central1-f zone”.

- **settings.nodes.parameters.sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **settings.nodes.parameters.withNATInstance**
  object

  Settings for the `WithNATInstance` layout.

  - **settings.nodes.parameters.withNATInstance.externalSubnetID**
    string

    If specified, an additional network interface will be added to the node (the latter will use it as a default route).

  - **settings.nodes.parameters.withNATInstance.internalSubnetCIDR**
    string

    CIDR of an automatically created subnet for the internal interface. Takes precedence over the `internalSubnetID` parameter.

  - **settings.nodes.parameters.withNATInstance.internalSubnetID**
    string

    ID of a subnet for the internal interface.

  - **settings.nodes.parameters.withNATInstance.natInstanceExternalAddress**
    string

    A reserved external IP address. When the `externalSubnetID` parameter is set, specify an address from that subnet.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **settings.nodes.parameters.withNATInstance.natInstanceInternalAddress**
    string

    Internal IP address of the NAT instance. To let Yandex Cloud assign the address, leave this parameter unset.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **settings.nodes.parameters.withNATInstance.natInstanceResources**
    object

    Computing resources that are allocated to the NAT instance. If not specified, the default values will be used.

    Warning. When these parameters change, the `terraform-manager` module restarts the NAT instance automatically if the `autoConvergerEnabled` parameter is set to `true`. The restart interrupts cluster network traffic for a short time.

    - **settings.nodes.parameters.withNATInstance.natInstanceResources.cores**
      integer

      Amount of CPU cores to provision on the NAT instance.

      Default: `2`

    - **settings.nodes.parameters.withNATInstance.natInstanceResources.memory**
      integer

Amount of primary memory in MB provision on the NAT instance.

Default: `2048`

- **settings.nodes.parameters.withNATInstance.natInstanceResources.platform**
  string

  Processor platform type on the NAT instance.

  Default: `standard-v2`

- **settings.nodes.parameters.zones**
  array of strings

  Limits the zones where the cloud provider places cluster nodes.

  You can list up to four zones. If the parameter is not set, Deckhouse Platform uses `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, and `ru-central1-e`.

  Instructions for using the `ru-central1-f` zone are provided in “Using the ru-central1-f zone”.

  - **Element of the array**
    string

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, `ru-central1-e`, `ru-central1-f`

- **settings.provider**
  object

  Required value

  Yandex Cloud connection settings.

  The provider reads the service account key from the `d8-credentials` Secret.

  - **settings.provider.parameters**
    object

    Required value

    Contains settings to connect to the Yandex Cloud API.

    - **settings.provider.parameters.cloudID**
      string

      Required value

      The cloud ID.

    - **settings.provider.parameters.folderID**
      string

      Required value

      ID of the directory.

- **settings.storage**
  object

  Storage subsystem settings.

  The subsystem provides the CSI driver and the StorageClasses for Yandex Cloud disk types.

  - **settings.storage.disabled**
    boolean

    Disables the storage subsystem.

    When set to `true`, disk provisioning in the cluster is unavailable.

    Default: `false`

  - **settings.storage.parameters**
    object

    Required value

    Parameters of the storage subsystem.

    - **settings.storage.parameters.excludedStorageClasses**
      array of strings

      List of StorageClasses to exclude from use in the cluster.

      Each item is a regular expression matched against the whole StorageClass name. For example, `network-ssd` excludes only the `network-ssd` StorageClass, and `network-ssd.*` excludes every StorageClass whose name starts with `network-ssd`.

      The list is applied after `provisionedStorageClasses`, so it also filters out the StorageClass defined there.

    - **settings.storage.parameters.provisionedStorageClasses**
      array of objects

Defines additional StorageClasses or overrides the parameters of the ones created by default.

A StorageClass created by default is overridden when its name matches the `name` field exactly.

- **settings.storage.parameters.provisionedStorageClasses.blockSize**
  string

  The block size of disks created by this StorageClass.

  If the parameter is omitted, the default Yandex Cloud block size is used — `4Ki`.

  The block size determines the maximum size of a network disk: for a `4Ki` block it is `8Ti`, and it doubles with each next increase of the block size — up to `256Ti` for a `128Ki` block.

  After a disk is created, its block size cannot be changed. Changing the `blockSize` parameter recreates the StorageClass, but does not change the block size of previously provisioned volumes.

  For details, see the official Yandex Cloud documentation.

  Allowed values: `4Ki`, `8Ki`, `16Ki`, `32Ki`, `64Ki`, `128Ki`

- **settings.storage.parameters.provisionedStorageClasses.name**
  string

  Required value

  The name of the StorageClass to create.

- **settings.storage.parameters.provisionedStorageClasses.type**
  string

  Required value

  The disk type.

  Allowed values: `network-hdd`, `network-ssd`, `network-ssd-nonreplicated`, `network-ssd-io-m3`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

> The module lifecycle stage: General Availability

YandexClusterConfiguration

Version: deckhouse.io/v1

> **Warning.** The resource is deprecated. The cloud provider configuration is set in the ModuleConfig resource of the `cloud-provider-yandex` module. Migrate the cluster to ModuleConfig as described in “How to migrate a cloud provider to ModuleConfig-based configuration”.

Describes the configuration of a cloud cluster in Yandex Cloud.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
nodeNetworkCIDR: 127.0.0.1/8
labels:
  label-2: b
sshPublicKey: "<SSH_PUBLIC_KEY>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
    coreFraction: 50
    externalIPAddresses:
    - 198.51.100.5
    - Auto
provider:
  cloudID: "<CLOUD_ID>"
  folderID: "<FOLDER_ID>"
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **dhcpOptions**
  object

  A list of DHCP parameters to use for all subnets.

  Note that setting dhcpOptions may lead to problems.

  - **dhcpOptions.domainName**
    string

    The name of the search domain.

  - **dhcpOptions.domainNameServers**
    array of strings

    A list of recursive DNS addresses.

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **existingNetworkID**
  string

  The ID of the existing VPC Network.

- **existingZoneToSubnetIDMap**
  object

  One or more pre-existing subnets mapped to respective zone.

  Warning. When using `cni-simple-bridge`, DP creates a route table that must be manually associated with the specified subnets. Only one route table can be associated with a subnet, so multiple clusters using `cni-simple-bridge` cannot be deployed in the same subnets. Starting with DP 1.76, new clusters in Yandex Cloud use `cni-cilium` in `VXLAN` mode by default. In this mode, pod traffic routing does not depend on Yandex Cloud route tables.

Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Example:

```
existingZoneToSubnetIDMap:
  ru-central1-a: e2lu8r1tbbtryhdpa9ro
  ru-central1-b: e2lu8r1tbbtryhdpa9ro
  ru-central1-d: e2lu8r1tbbtryhdpa9ro
  ru-central1-e: e2lu8r1tbbtryhdpa9ro
  ru-central1-f: e2lu8r1tbbtryhdpa9ro
```

- **kind**
  string

  Required value

  Allowed values: `YandexClusterConfiguration`

- **labels**
  object

  Labels to attach to resources created in the Yandex Cloud.

  Note that you have to re-create all the machines to add new labels if labels were modified in the running cluster.

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

  Allowed values: `Standard`, `WithoutNAT`, `WithNATInstance`

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the YandexInstanceClass.

    - **masterNodeGroup.instanceClass.additionalLabels**
      object

      Additional labels.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **masterNodeGroup.instanceClass.cores**
      integer

      Required value

      Amount of CPU cores to provision on a Yandex Compute Instance.

    - **masterNodeGroup.instanceClass.diskSizeGB**
      integer

      Yandex Compute Instance disk size in gibibytes.

      Default: `50`

      Example:

      ```
      diskSizeGB: 0
      ```

    - **masterNodeGroup.instanceClass.diskType**
      string

      Yandex Compute Instance disk type. Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

Default: `network-ssd`

Allowed values: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

Example:

```
diskType: network-ssd-io-m3
```

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  etcd disk size in gibibytes.

  If this parameter is changed, each etcd disk must be manually expanded in the Yandex Cloud interface (the reason for this behavior is in the issue).

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.externalIPAddresses**
  array of strings

  A list of external IP addresses, listed in the order of the zones where nodes will be created.

  The following values can be specified in the list:

  - IP address from an additional external network for the corresponding zone (parameter `externalSubnetIDs`);
  - reserved public IP address, if the list of additional external networks is not defined (parameter `externalSubnetIDs`);
  - `Auto`, to order a public IP address in the corresponding zone.

  Please note:

  - The array elements must be listed according to the order of zones (parameter `zones`) in which the nodes will be created.

    For example:

    ```
    - <RU-CENTRAL1-A-IP-ADDRESS>
    - <RU-CENTRAL1-B-IP-ADDRESS>
    - <RU-CENTRAL1-D-IP-ADDRESS>
    ```

  - The number of elements in the `externalIPAddresses` array must equal the number of nodes in the group (the `replicas` parameter). To automatically order public IP addresses in the zone, use the value `Auto` in the list of the `externalIPAddresses` parameter at the corresponding position (matching the order of zones).
  - If `externalIPAddresses` is not specified, the nodes in the group will be ordered without external IP addresses.
  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **masterNodeGroup.instanceClass.externalSubnetID**
  Deprecated
  string

  If specified, an additional network interface will be added to the node (the latter will use it as a default route).

- **masterNodeGroup.instanceClass.externalSubnetIDs**
  array of strings

  List of additional external networks connected to the nodes.

  The node interface that is in the list will be assigned as the default gateway.

  Also, a route for the node’s internal network interface will be added for the entire subnet specified in nodeNetworkCIDR.

  The array elements **must** be listed in the same order as the zones (parameter `zones`) in which the nodes will be created. For example:

  ```
  - <RU-CENTRAL1-A-SUBNET-ID>
  - <RU-CENTRAL1-B-SUBNET-ID>
  - <RU-CENTRAL1-D-SUBNET-ID>
  ```

- **masterNodeGroup.instanceClass.imageID**
  string

  Required value

  Image ID to use while provisioning Yandex Compute Instances.

Example:

```
imageID: fd83ica41cade1mj35sr
```

- **masterNodeGroup.instanceClass.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **masterNodeGroup.instanceClass.networkType**
  string

  Network type.

  Default: `Standard`

  Allowed values: `Standard`, `SoftwareAccelerated`

- **masterNodeGroup.instanceClass.platform**
  string

  The type of virtual machine to create.

  Default: `standard-v2`

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array

  Not required value.

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, `ru-central1-e`, `ru-central1-f`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the YandexInstanceClass.

    - **nodeGroups.instanceClass.additionalLabels**
      object

      Additional labels.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **nodeGroups.instanceClass.coreFraction**
      integer

      Percent of reserved CPU capacity on a Yandex Compute Instance. Details...

      Default: `100`

      Allowed values: `5`, `20`, `50`, `100`

      Example:

```
coreFraction: 20
```

- **nodeGroups.instanceClass.cores**
  integer

  Required value

  Amount of CPU cores to provision on a Yandex Compute Instance.

- **nodeGroups.instanceClass.diskSizeGB**
  integer

  Yandex Compute Instance disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGB: 0
  ```

- **nodeGroups.instanceClass.diskType**
  string

  Yandex Compute Instance disk type. Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

  Default: `network-ssd`

  Allowed values: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

  Example:

  ```
  diskType: network-ssd-io-m3
  ```

- **nodeGroups.instanceClass.externalIPAddresses**
  array of strings

  A list of external IP addresses, listed in the order of the zones where nodes will be created.

  The following values can be specified in the list:

  - IP address from an additional external network for the corresponding zone (parameter `externalSubnetIDs`);
  - reserved public IP address, if the list of additional external networks is not defined (parameter `externalSubnetIDs`);
  - `Auto`, to order a public IP address in the corresponding zone.

  Please note:

  - The array elements must be listed according to the order of zones (parameter `zones`) in which the nodes will be created.

    For example:

    ```
    - <RU-CENTRAL1-A-IP-ADDRESS>
    - <RU-CENTRAL1-B-IP-ADDRESS>
    - <RU-CENTRAL1-D-IP-ADDRESS>
    ```

  - The number of elements in the `externalIPAddresses` array must equal the number of nodes in the group (the `replicas` parameter). To automatically order public IP addresses in the zone, use the value `Auto` in the list of the `externalIPAddresses` parameter at the corresponding position (matching the order of zones).
  - If `externalIPAddresses` is not specified, the nodes in the group will be ordered without external IP addresses.
  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **nodeGroups.instanceClass.externalSubnetID**
  Deprecated
  string

  If specified, an additional network interface will be added to the node (the latter will use it as a default route).

- **nodeGroups.instanceClass.externalSubnetIDs**
  array of strings

  List of additional external networks connected to the nodes.

  The node interface that is in the list will be assigned as the default gateway.

  Also, a route for the node's internal network interface will be added for the entire subnet specified in nodeNetworkCIDR.

  The array elements **must** be listed in the same order as the zones (parameter `zones`) in which the nodes will be created. For example:

```
- <RU-CENTRAL1-A-SUBNET-ID>
- <RU-CENTRAL1-B-SUBNET-ID>
- <RU-CENTRAL1-D-SUBNET-ID>
```

- **nodeGroups.instanceClass.imageID**
  string

  Required value

  Image ID to use while provisioning Yandex Compute Instances.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.networkType**
  string

  Network type.

  Default: `Standard`

  Allowed values: `Standard`, `SoftwareAccelerated`

- **nodeGroups.instanceClass.platform**
  string

  The type of virtual machine to create.

  Default: `standard-v2`

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

> **i** Available fields: `effect`, `key`, and `values`.

Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string
- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.
- **nodeGroups.zones**
  array

  A limited set of zones in which nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, `ru-central1-e`, `ru-central1-f`

- **nodeNetworkCIDR**
  string

  Required value

  IP address range for node subnets.

  Deckhouse Platform divides the range into four equal parts:

  - `ru-central1-a` uses the first part.
  - `ru-central1-b` uses the second part.
  - `ru-central1-e` uses the third part.
  - `ru-central1-d` uses the fourth part.

  Instructions for using the `ru-central1-f` zone are provided in “Using the ru-central1-f zone”.
- **provider**
  object

  Required value

  Contains settings to connect to the Yandex Cloud API.

  - **provider.cloudID**
    string

    Required value

    The cloud ID.
  - **provider.folderID**
    string

    Required value

    ID of the directory.
  - **provider.serviceAccountJSON**
    string

    Required value

    A key to the Service Account in the JSON format.

    You can get it by using `[yc iam key create](environment.html)` command.

    Pattern: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

    Example:

```
serviceAccountJSON: |
  {
    "id": "...",
    "service_account_id": "...",
    "created_at": "2022-08-04T05:38:34.756137618Z",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----...-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----...-----END PRIVATE KEY-----\n"
  }
```

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **withNATInstance**
  object

  Settings for the `WithNATInstance` layout.

  - **withNATInstance.exporterAPIKey**
    string

    API-key for cloud metrics exporter.

    - If parameter is empty, cloud metrics exporter will not be deployed in the cluster.
    - If parameter is `Auto`, Deckhouse will create service account with the `monitoring.viewer` role and create API-key manually. Provider service account should have the `admin` role.
    - Any other value is considered a valid API-key. See this instruction for creating API-key. Service account should have `monitoring.viewer` role.

    Default:

  - **withNATInstance.externalSubnetID**
    string

    If specified, an additional network interface will be added to the node (the node will use it as a default route).

  - **withNATInstance.internalSubnetCIDR**

    CIDR of an automatically created subnet for the internal interface. Overrides `internalSubnetID` parameter.

  - **withNATInstance.internalSubnetID**
    string

    ID of a subnet for the internal interface.

  - **withNATInstance.natInstanceExternalAddress**
    string

    A reserved external IP address (or `externalSubnetID` address if specified).

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **withNATInstance.natInstanceInternalAddress**
    Deprecated
    string

    Consider using automatically generated address instead.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **withNATInstance.natInstanceResources**
    object

    Computing resources that are allocated to the NAT instance. If not specified, the default values will be used.

    > **i** **Warning.** If these parameters are changed, `terraform-auto-converger` will automatically restart NAT-instance if autoConvergerEnabled is set to `true`. This may result in a temporary interruption of network traffic in the cluster.

    - **withNATInstance.natInstanceResources.cores**
      integer

      Amount of CPU cores to provision on the NAT instance.

      Default: `2`

    - **withNATInstance.natInstanceResources.memory**
      integer

      Amount of primary memory in MB provision on the NAT instance.

      Default: `2048`

    - **withNATInstance.natInstanceResources.platform**

string

Processor platform type on the NAT instance.

Default: `standard-v2`

- **zones**
  array

  Limits the zones that the cloud provider works with.

  You can list up to four zones. If the parameter is not set, Deckhouse Platform uses `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, and `ru-central1-e`.

  Instructions for using the `ru-central1-f` zone are provided in “Using the ru-central1-f zone”.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, `ru-central1-e`, `ru-central1-f`

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

The module lifecycle stage: General Availability

To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Creating a service account

You need to create a service account with the editor role with the cloud provider so that Deckhouse Platform (DP) can manage cloud resources. The detailed instructions for creating a service account with Yandex Cloud are available in the provider’s documentation. Below, we will provide a brief overview of the necessary actions:

1. Create a user named `deckhouse`:

   ```
   yc iam service-account create --name deckhouse
   ```

   The command output will contain its parameters:

   ```
   id: <USER_ID>
   folder_id: <FOLDER_ID>
   created_at: "YYYY-MM-DDTHH:MM:SSZ"
   name: deckhouse
   ```

2. Assign the required roles to the newly created user for your cloud:

```
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role compute.editor --subject serviceAccount:
<USER_ID>
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role vpc.admin --subject serviceAccount:<USER_ID>
yc resource-manager folder add-access-binding --id <FOLDER_ID> --role load-balancer.editor --subject
serviceAccount:<USER_ID>
```

3. Create a JSON file containing the parameters for user authorization in the cloud. These parameters will be used to log in to the cloud:

```
yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
```

> **i** A new cluster may require increased quotas. If necessary, also reserve a public IP address as described in “Reserving a public IP address”.

Credentials Secret

The service account key is stored in a separate Secret rather than in ModuleConfig. The provider reads it when starting the components that access the Yandex Cloud API.

The Secret is created during cluster installation together with the other resources of the initial configuration. If the provider is connected to a running cluster, the Secret is applied after the module creates the namespace. The Secret must meet the following requirements:

- The name is `d8-credentials` and the namespace is `d8-cloud-provider-yandex`.
- The type is `cloud-provider.deckhouse.io/credentials`.
- The `authScheme` field is set to `serviceAccount`.
- The `secret` field contains the service account key in JSON.

```
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-yandex
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: serviceAccount
  secret: |
    <SERVICE_ACCOUNT_JSON>
```

Replace `<SERVICE_ACCOUNT_JSON>` with the contents of the `deckhouse-sa-key.json` file.

To change the credentials, update the `secret` field:

```
d8 k -n d8-cloud-provider-yandex edit secret d8-credentials
```

In the “WithNATInstance layout”, the API key for monitoring the NAT instance is stored in a second Secret, `d8-credentials-exporter`, in the same namespace. Its `authScheme` field is set to `apiToken` and the key is placed in the `secret` field.

In clusters migrated to ModuleConfig, the service account key moves to the `d8-credentials` Secret from the `provider.serviceAccountJSON` parameter of the YandexClusterConfiguration resource. The monitoring API key moves to the `d8-credentials-exporter` Secret from the `withNATInstance.exporterAPIKey` parameter.

Quotas

> **i** Note that you need to increase the quotas using the Yandex console when provisioning a new cluster.

Recommended quotas for a new cluster:

- The number of virtual processors: 64.
- The total volume of SSD disks: 2000 GB.
- The number of virtual machines: 25.
- The total amount of RAM of virtual machines: 256 GB.

Yandex Cloud integration

> **⚠** Starting with DP version 1.77, Yandex Cloud uses the `cilium` CNI by default for new clusters. Existing clusters keep the current CNI configuration.
>
> New clusters require Linux kernel version 5.8 or newer on all nodes. Make sure firewalls or security groups allow inter-node UDP traffic for Cilium VXLAN. For details, see the installation requirements, “Network interaction of the platform components”, and the `cni-cilium` module documentation.

Configuring security groups

For every cloud network, Yandex Cloud creates a default security group, including for the cluster network. That group allows any inbound and outbound traffic and applies to the virtual machine network interfaces that are not assigned another group.

You can restrict the cluster traffic with the rules of that group. Assigning another security group to the nodes is not possible, as neither the module settings nor the YandexInstanceClass resource have parameters for that.

⚠ Do not assign security groups to the nodes manually in the Yandex Cloud console. The `dhctl converge` command brings the virtual machines to the state described in the cluster configuration and may remove such an assignment.

✖ Do not delete the rules that allow any traffic until you have configured the other rules of the group. Otherwise, the group blocks the traffic between the nodes and the cluster components lose connection to each other.

Errors in the rules disrupt the cluster. Before configuring the rules in a production cluster, consult the security group usage details in Yandex Cloud.

If the cluster uses `cilium` in VXLAN mode, make sure the security group allows inter-node UDP traffic on the ports required for Cilium. These ports are listed in “Network interaction of the platform components”.

To configure the rules of the security group:

1. Find out in which cloud network the Deckhouse Platform cluster is running.

   The network name matches the `prefix` parameter of the ModuleConfig `global`. You can retrieve it using the following command:

   ```
   d8 k get mc global -o jsonpath='{.spec.settings.prefix}{"\n"}'
   ```

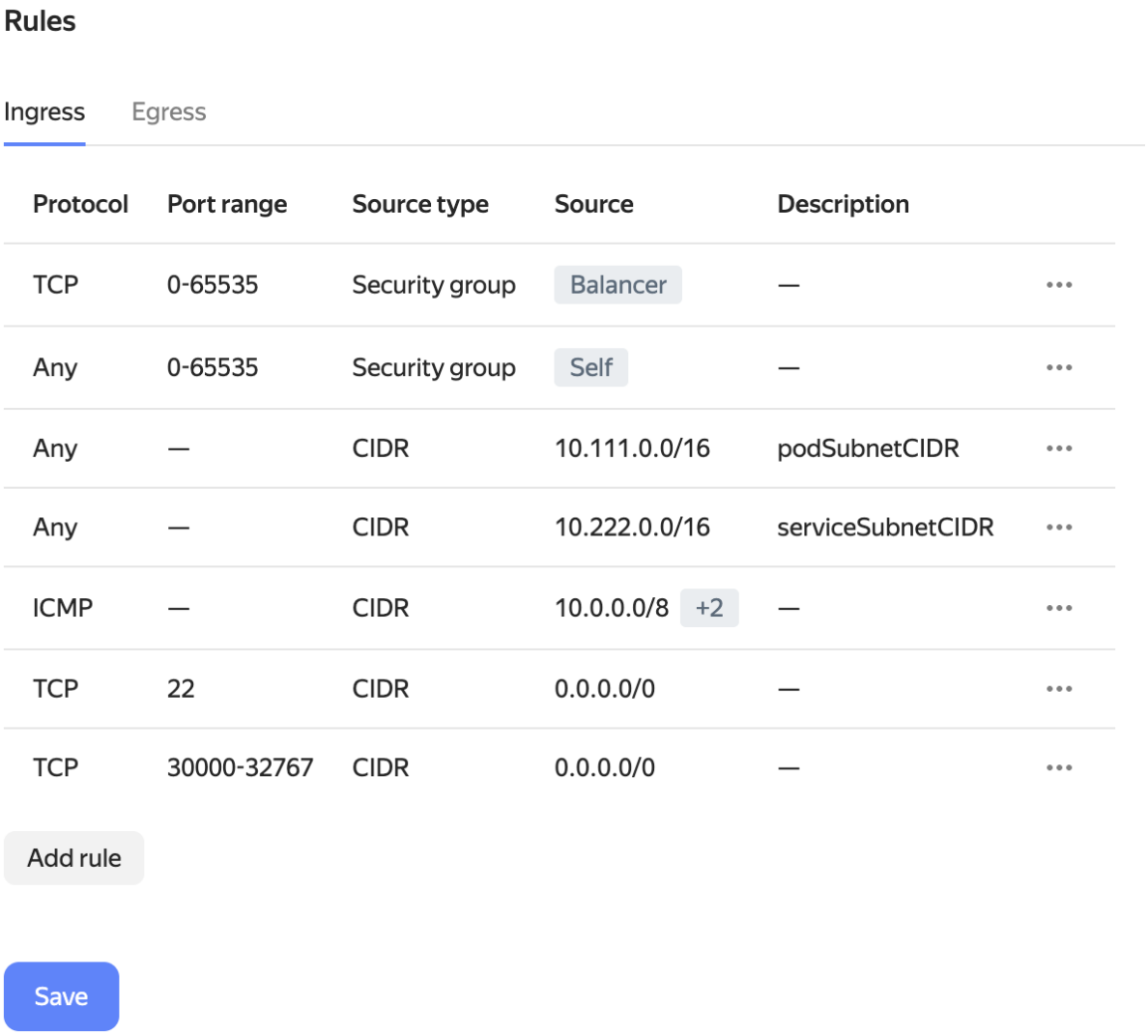
   If the value is empty, get it from the deprecated `cloud.prefix` parameter of ClusterConfiguration (the prefix has moved to the ModuleConfig `global`):

   ```
   d8 k get secrets -n kube-system d8-cluster-configuration -ojson | \
     jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
   ```

2. In the Yandex Cloud console, select the Virtual Private Cloud service and navigate to the “Security Groups” section. You should see a single security group labeled `Default`.

3. Create rules as described in Yandex Cloud instructions.



4. Delete the rule that allows for any **inbound** traffic (in the screenshot above it has already been deleted), and save the changes.

Yandex Lockbox integration

The External Secrets Operator allows you to synchronize Yandex Lockbox secrets with the Deckhouse Platform cluster secrets.

The instructions below are meant to be viewed as a *Quick Start* guide. To use integration in production environments, please review the following resources:

- Yandex Lockbox
- Synchronizing with Yandex Lockbox secrets
- External Secrets Operator

Deployment instructions

1. Create a service account required for the External Secrets Operator:

   ```
   yc iam service-account create --name eso-service-account
   ```

2. Create an authorized key for the service account and save it to a file:

   ```
   yc iam key create --service-account-name eso-service-account --output authorized-key.json
   ```

3. Assign `lockbox.editor`, `lockbox.payloadViewer`, and `kms.keys.encrypterDecrypter` roles to a service account to access all catalog secrets:

```
folder_id=<FOLDER_ID>
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.editor
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.payloadViewer
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role kms.keys.encrypterDecrypter
```

For advanced customization, check out access control in Yandex Lockbox.

4. Install the External Secrets Operator using the Helm chart according to instructions.

   Note that you may need to set `nodeSelector`, `tolerations` and other parameters. To do this, use the `./external-secrets/values.yaml` file after unpacking the Helm chart.

   Pull and extract the chart:

   ```
   helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
     --version 0.5.5 \
     --untar
   ```

   Install the Helm chart:

   ```
   helm install -n external-secrets --create-namespace \
     --set-file auth.json=authorized-key.json \
     external-secrets ./external-secrets/
   ```

   Where:

   - `authorized-key.json` — the name of the file with the authorized key from step 2.

5. Create a SecretStore with the `sa-creds` secret in it:

   ```
   d8 k -n external-secrets apply -f - <<EOF

   apiVersion: external-secrets.io/v1alpha1
   kind: SecretStore
   metadata:
     name: secret-store
   spec:
     provider:
       yandexlockbox:
         auth:
           authorizedKeySecretRef:
             name: sa-creds
             key: key
   EOF
   ```

   Where:

   - `sa-creds` — the name of the Secret that contains the authorized key. This secret should show up after the Helm chart has been installed.
   - `key` — the name of the key in the `.data` field of the secret above.

Checking that everything works as expected

1. Check the status of the External Secrets Operator:

   ```
   d8 k -n external-secrets get po
   ```

   An example of the output:

   ```
   NAME                                                READY   STATUS    RESTARTS   AGE
   external-secrets-55f78c44cf-dbf6q                   1/1     Running   0          77m
   external-secrets-cert-controller-78cbc7d9c8-rszhx   1/1     Running   0          77m
   external-secrets-webhook-6d7b66758-s7v9c            1/1     Running   0          77m
   ```

2. Check the state of the secrets store you created:

   ```
   d8 k -n external-secrets get secretstores.external-secrets.io
   ```

   An example of the output:

```
NAME           AGE   STATUS
secret-store   69m   Valid
```

3. Create a Yandex Lockbox secret with the following parameters:
   - **Name** — `lockbox-secret`.
   - **Key** — enter the non-confidential identifier `password`.
   - **Value** — enter the confidential data to store `p@$$w0rd`.
4. Create an ExternalSecret object that refers to the `lockbox-secret` secret in the `secret-store`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
  - secretKey: password
    remoteRef:
      key: <SECRET_ID>
      property: password
EOF
```

   Where:

   - `spec.target.name` — the name of the new secret. The External Secrets Operator will create this secret in the Deckhouse Platform cluster and populate it with the parameters of the Yandex Lockbox's `lockbox-secret`.
   - `spec.data[].secretKey` — the name of the key in the `.data` field of the secret that the External Secrets Operator will create.
   - `spec.data[].remoteRef.key` — identifier of the Yandex Lockbox's `lockbox-secret` created earlier, e.g., `e6q28nvfmhu539******`.
   - `spec.data[].remoteRef.property` — the **key** you specified earlier for the Yandex Lockbox's `lockbox-secret`.
5. Make sure that the new `k8s-secret` key contains the `lockbox-secret` value:

```
d8 k -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

   The output of the command should contain the **value** of the `password` key of the `lockbox-secret` created earlier:

```
p@$$w0rd
```

Yandex Managed Service for Prometheus integration

This integration lets you use the Yandex Managed Service for Prometheus as an external metrics repository, e.g., for long-term metrics storage.

Writing metrics

1. Create a service account with the `monitoring.editor` role.
2. Create an API key for the service account.
3. Create a PrometheusRemoteWrite resource:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <URL_TO_WRITE_METRICS>
  bearerToken: <API_KEY>
EOF
```

   Where:

   - `<URL_TO_WRITE_METRICS>` — URL from the Yandex Monitoring/Prometheus/Writing Metrics page.
   - `<API_KEY>` — the API key you created in the previous step, e.g.,

`AQVN1HHJRSrfo9jU3aopsXrJyfq_UHs********`.

You may also specify additional parameters; refer to the documentation.

More details about this feature can be found in Yandex Cloud documentation.

Reading metrics with Grafana

1. Create a service account with the `monitoring.viewer` role.
2. Create an API key for the service account.
3. Create a GrafanaAdditionalDatasource resource:

```
d8 k apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <URL_READING_METRICS_WITH_GRAFANA>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_KEY>
EOF
```

Where:

- `<URL_READING_METRICS_WITH_GRAFANA>` — URL from the Yandex Monitoring/Prometheus/Reading Metrics with Grafana page.
- `<API_KEY>` — the API key you created in the previous step, e.g., `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`.

You may also specify additional parameters; refer to the documentation.

More details about this feature can be found in Yandex Cloud documentation.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

YandexInstanceClass

Scope: Cluster

- v1
- v1alpha1

- **metadata**
  object
- **spec**
  object

  Required value

  YandexInstanceClassSpec defines the desired state of the YandexInstanceClass.

  - **spec.additionalLabels**
    object

    Additional labels to be attached to the Yandex Compute Instance resource.

Example:

```
additionalLabels: "{project: cms-production, severity: critical}"
```

- **spec.additionalSubnets**
  array of strings

  Subnet IDs that VirtualMachines' secondary NICs will connect to.

  For `CloudEphemeral` nodes, every subnet in the list is attached as a separate network interface.

  For `CloudPermanent` nodes, a single subnet is selected from the list by the node index, so the list must contain at least as many subnets as there are nodes in the group.

  Example:

  ```
  additionalSubnets: "[b0csh41c1or82vuch89v, e2lgddi5svochh5fbq96]"
  ```

- **spec.assignPublicIPAddress**
  boolean

  Should a public external IPv4 address be assigned to a provisioned Yandex Compute Instance.

  Default: `false`

  Example:

  ```
  assignPublicIPAddress: 'false'
  ```

- **spec.coreFraction**
  integer

  Percent of reserved CPU capacity on a Yandex Compute Instance.

  More about performance levels.

  Default: `100`

  Allowed values: `5`, `20`, `50`, `100`

  Example:

  ```
  coreFraction: '20'
  ```

- **spec.cores**
  integer

  Required value

  Amount of CPU cores to provision on a Yandex Compute Instance.

  Example:

  ```
  cores: '4'
  ```

- **spec.diskSizeGB**
  integer

  Yandex Compute Instance disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGB: '20'
  ```

- **spec.diskType**
  string

  Instance disk type.

  Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

  Default: `"network-hdd"`

  Allowed values: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`, `network-ssd-io-m3`

  Example:

  ```
  diskType: network-hdd
  ```

- **spec.etcdDiskSizeGB**
  integer

  Disk size for etcd is set only for InstanceClass intended for master NodeGroup. The value is specified in `GiB`.

  Example:

```
etcdDiskSizeGB: '20'
```

- **spec.gpus**
  integer

  Number of GPUs on a Yandex Compute Instance.

  Default: `0`

  Example:

  ```
  gpus: '4'
  ```

- **spec.imageID**
  string

  Image ID to use while provisioning Yandex Compute Instances.

  If the parameter is not set, the image of the instance class of the `master` node group is used. In a cluster that has not been migrated to the ModuleConfig settings, the `masterNodeGroup.instanceClass.imageID` parameter of the YandexClusterConfiguration resource is used instead.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **spec.mainSubnet**
  string

  Subnet ID that VirtualMachines’ primary NIC will connect to.

  If the parameter is not specified, Deckhouse Platform selects the network. When the `existingZoneToSubnetIDMap` parameter is set, the network is taken from it, otherwise the network created during the cluster deployment is used.

  Example:

  ```
  mainSubnet: e9bnc7g9mu9mper9clk4
  ```

- **spec.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: '8192'
  ```

- **spec.networkType**
  string

  Network type: `Standard` or `SoftwareAccelerated`.

  Default: `Standard`

  Allowed values: `Standard`, `SoftwareAccelerated`

- **spec.platformID**
  string

  Platform ID.

  List of available platforms...

  Default: `standard-v3`

- **spec.preemptible**
  boolean

  Should a provisioned Yandex Compute Instance be preemptible.

  For more information about preemptible virtual machines, read the provider’s documentation.

  Default: `false`

- **status**
  object

  YandexInstanceClassStatus stores information about the YandexInstanceClass resource.

  - **status.nodeGroupConsumers**
    array of strings

    NodeGroupConsumers lists the names of NodeGroups that use this instance class.

- **metadata**

object

- **spec**
  object

  Required value

  InstanceClassSpec defines the desired state of the YandexInstanceClass.

  - **spec.additionalLabels**
    object

    Additional labels to be attached to the Yandex Compute Instance resource.

    Example:

    ```
    additionalLabels: "{project: cms-production, severity: critical}"
    ```

  - **spec.additionalSubnets**
    array of strings

    Subnet IDs that VirtualMachines' secondary NICs will connect to.

    For `CloudEphemeral` nodes, every subnet in the list is attached as a separate network interface.

    For `CloudPermanent` nodes, a single subnet is selected from the list by the node index, so the list must contain at least as many subnets as there are nodes in the group.

    Example:

    ```
    additionalSubnets: "[b0csh41c1or82vuch89v, e2lgddi5svochh5fbq96]"
    ```

  - **spec.assignPublicIPAddress**
    boolean

    Should a public external IPv4 address be assigned to a provisioned Yandex Compute Instance.

    Default: `false`

    Example:

    ```
    assignPublicIPAddress: 'false'
    ```

  - **spec.coreFraction**
    integer

    Percent of reserved CPU capacity on a Yandex Compute Instance.

    More about performance levels.

    Default: `100`

    Allowed values: `5`, `20`, `50`, `100`

    Example:

    ```
    coreFraction: '20'
    ```

  - **spec.cores**
    integer

    Required value

    Amount of CPU cores to provision on a Yandex Compute Instance.

    Example:

    ```
    cores: '4'
    ```

  - **spec.diskSizeGB**
    integer

    Yandex Compute Instance disk size in gibibytes.

    Default: `20`

    Example:

    ```
    diskSizeGB: '50'
    ```

  - **spec.diskType**
    string

    Instance disk type.

    Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

    For more information about possible disk types, read the provider's documentation.

Default: `"network-hdd"`

Allowed values: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`

Example:

```
diskType: network-hdd
```

- **spec.gpus**
integer

  Number of GPUs on a Yandex Compute Instance.

  Default: `0`

  Example:

  ```
  gpus: '4'
  ```

- **spec.imageID**
string

  Image ID to use while provisioning Yandex Compute Instances.

  If the parameter is not set, the image of the instance class of the `master` node group is used. In a cluster that has not been migrated to the ModuleConfig settings, the `masterNodeGroup.instanceClass.imageID` parameter of the YandexClusterConfiguration resource is used instead.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **spec.mainSubnet**
string

  Subnet ID that VirtualMachines’ primary NIC will connect to.

  If the parameter is not specified, Deckhouse Platform selects the network. When the `existingZoneToSubnetIDMap` parameter is set, the network is taken from it, otherwise the network created during the cluster deployment is used.

  Example:

  ```
  mainSubnet: e9bnc7g9mu9mper9clk4
  ```

- **spec.memory**
integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: '8192'
  ```

- **spec.networkType**
string

  Network type: STANDARD or SOFTWARE_ACCELERATED

  Default: `STANDARD`

  Allowed values: `STANDARD`, `SOFTWARE_ACCELERATED`

- **spec.platformID**
string

  Platform ID.

  List of available platforms...

  Default: `standard-v2`

- **spec.preemptible**
boolean

  Should a provisioned Yandex Compute Instance be preemptible.

  For more information about preemptible virtual machines, read the provider’s documentation.

  Default: `false`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

Setting up the INTERNAL LoadBalancer

Attach the following annotation to the service:

```
yandex.cpi.flant.com/listener-subnet-id: SubnetID
```

The annotation links the LoadBalancer with the appropriate Subnet.

Using a separate target group for a NodeGroup

By default, Yandex Cloud Controller Manager adds all suitable cluster nodes to the default target group. To assign nodes to a separate target group, use the `yandex.cpi.flant.com/target-group-name-prefix` annotation. The annotation is described in the “Service annotations” section.

1. In the NodeGroup whose nodes should be included in a separate target group, specify the `yandex.cpi.flant.com/target-group-name-prefix` annotation in the `spec.nodeTemplate.annotations` parameter. For example:

   ```
   spec:
     nodeTemplate:
       annotations:
         yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation is propagated to the Node objects. Yandex Cloud Controller Manager adds all suitable nodes carrying the same annotation value to the target group, regardless of their NodeGroup.

2. In the Service object of the LoadBalancer type, specify the same annotation with the same value. For example:

   ```
   metadata:
     annotations:
       yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation on the Service identifies the target group used by the LoadBalancer but does not select nodes.

3. Make sure that the values of `yandex.cpi.flant.com/target-group-name-prefix` in the NodeGroup and Service match. In the example above, both resources use the `frontend-` value.

4. The target group will be created with a name generated according to the following pattern:

   ```
   <ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
   ```

   For example, if the annotation value is `frontend-`, the cluster name is `my-cluster-`, and the network ID is `enp123456789`, the target group name will be:

   ```
   frontend-my-cluster-enp123456789
   ```

Configuration example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer
```

> ⚠ Yandex Cloud does not allow a single target, identified by the (`SubnetID, IP`) pair, to belong to multiple target groups at the same time.
>
> When the target group is changed, Yandex Cloud Controller Manager first removes the target from the current group and then adds it to the new one. This transition may cause a brief interruption in traffic handling.

Nodes with a dedicated target group prefix are excluded from the default target group. As a result, load balancers that use the default target group stop routing traffic to those nodes.

This automatic transition is performed only between target groups managed by Yandex Cloud Controller Manager. If a target already belongs to another target group that is not managed by Yandex Cloud Controller Manager, remove it from that group before using the annotation.

After the prefix is changed or removed, the previous target group may remain empty. If it is no longer used by any load balancers, delete it manually.

Reserving a public IP address

This on is used in `externalIPAddresses` and `natInstanceExternalAddress`. It also can be used for a bastion host.

```
$ yc vpc address create --external-ipv4 zone=ru-central1-a
id: e9b4cfmmnc1mhgij75n7
folder_id: b1gog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
  requirements: {}
reserved: true
```

dhcpOptions-related problems and ways to address them

Using DNS servers that differ from those provided by Yandex Cloud in the DHCP settings is a temporary solution. It will be abandoned after Yandex Cloud will introduce the Managed DNS service. To get around the restrictions described below, we recommend using `stubZones` from the `kube-dns` module.

Editing parameters

Pay attention to the following nuances:

1. When changing parameters, you need to invoke `netplan apply` or a similar command that forces the update of the DHCP lease.
2. You will need to restart all hostNetwork Pods (especially `kube-dns`) for the new `resolv.conf` settings to take effect.

Aspects of the use

If the dhcpOptions parameter is set, all DNS are routed to the DNS servers specified. These DNS servers **must** serve DNS requests to the Internet and (if needed) resolve intranet resources.

Do not use this option if the recursive DNSs specified cannot resolve the same list of zones that the recursive DNSs in the Yandex Cloud subnet can resolve.

Changing `networkType` does not recreate CloudEphemeral nodes

In clusters that use Machine Controller Manager (MCM), changing the `networkType` parameter in a YandexInstanceClass resource does not trigger automatic recreation of existing CloudEphemeral nodes. Although the MachineClass is updated, the network acceleration type of existing virtual machines in Yandex Cloud remains unchanged.

DP intentionally does not take the `networkType` parameter into account when determining whether CloudEphemeral nodes should be recreated in MCM. If it did, updating DP would recreate CloudEphemeral nodes in all clusters where `networkType` is specified, even if its value had not changed.

In Cluster API, the `networkType` parameter is taken into account when determining whether nodes should be recreated, so changing it automatically triggers node recreation. New CloudEphemeral NodeGroups in Yandex Cloud use Cluster API by default.

If you changed `networkType` in an MCM-based cluster and want the new value to apply to existing CloudEphemeral nodes, recreate them manually — instructions in the documentation.

Setting a custom StorageClass as default

Specify the StorageClass name in the global `defaultClusterStorageClass` parameter. After that, DP removes the `storageclass.kubernetes.io/is-default-class='true'` annotation from the StorageClass that was set as the default one before.

```
d8 k edit mc global
```

Resizing a PVC

You can increase the size of an existing PVC by changing the value of the `spec.resources.requests.storage` parameter, without stopping or recreating the pod that uses it.

After the value of `spec.resources.requests.storage` is changed, the CSI driver performs the following operations:

- increases the size of the disk in Yandex Cloud;
- updates the size of the associated PersistentVolume;
- expands the file system on the node to which the volume is attached.

During the operation, the pod continues running, and the mounted volume remains available to the application. After the resize operation is complete, the new file system size becomes available inside the container without restarting the pod.

> **i** Decreasing the size of a PVC is not supported.

To increase the size of a PVC, follow these steps:

1. Get the name of the StorageClass used by the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME> \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the PVC;
   - `<PVC_NAME>` — the name of the PVC to resize.

   For example:

   ```
   d8 k -n production get pvc application-data \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Example output:

   ```
   network-ssd
   ```

   In this example, the `application-data` PVC uses the `network-ssd` StorageClass.

2. Make sure that the StorageClass allows volume expansion:

   ```
   d8 k get storageclass <STORAGECLASS_NAME> \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Where `<STORAGECLASS_NAME>` is the name of the StorageClass obtained in the previous step.

   For example:

   ```
   d8 k get storageclass network-ssd \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Example output:

   ```
   true
   ```

3. Check the current status and size of the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production get pvc application-data
   ```

   Example output:

   ```
   NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
   STORAGECLASS
   application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
   network-ssd
   ```

   Make sure that:

   - the PVC is in the `Bound` state;
   - the `CAPACITY` field shows the current PVC size;
   - the `STORAGECLASS` field shows the StorageClass checked in the previous step.

4. Increase the PVC size:

   ```
   d8 k -n <NAMESPACE> edit pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production edit pvc application-data
   ```

Specify the new PVC size in the `spec.resources.requests.storage` field:

```
spec:
  resources:
    requests:
      storage: 30Gi
```

In this example, the PVC size is increased to `30Gi`.

Save the changes and close the editor.

> ⚠ For the `network-ssd-nonreplicated` and `network-ssd-io-m3` StorageClasses, the size must be a multiple of 93Gi.

5. Wait for the PVC resize operation to complete:

```
d8 k -n <NAMESPACE> get pvc <PVC_NAME> --watch
```

Where:

- `<NAMESPACE>` — the namespace containing the PVC;
- `<PVC_NAME>` — the name of the PVC being resized.

For example:

```
d8 k -n production get pvc application-data --watch
```

During the resize operation, the `CAPACITY` field may continue to show the previous size:

```
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
network-ssd
```

The operation is complete when the `CAPACITY` field shows the new PVC size:

```
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   30Gi       RWO
network-ssd
```

6. Check the PVC events:

```
d8 k -n <NAMESPACE> describe pvc <PVC_NAME>
```

For example:

```
d8 k -n production describe pvc application-data
```

The following events may appear during the resize operation:

```
ExternalExpanding
Resizing
FileSystemResizeRequired
```

The following event indicates that the file system was successfully expanded:

```
FileSystemResizeSuccessful
```

For example:

```
Normal  FileSystemResizeSuccessful  kubelet  MountVolume.NodeExpandVolume succeeded for volume
"pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
```

7. Check the file system size inside the pod:

```
d8 k -n <NAMESPACE> exec <POD_NAME> -- \
  df -hT <MOUNT_PATH>
```

Where:

- `<NAMESPACE>` — the namespace containing the pod;
- `<POD_NAME>` — the name of the pod using the PVC;
- `<MOUNT_PATH>` — the path inside the container where the PVC is mounted.

For example:

```
d8 k -n production exec application-0 -- \
  df -hT /data
```

Example output:

```
Filesystem   Type   Size    Used   Avail   Use%   Mounted on
/dev/vde     ext4   29.4G   22M    29.4G    1%    /data
```

> **i** The actual file system size may be slightly smaller than the PVC size due to file system metadata.

Adding CloudStatic nodes to a cluster

For VMs that you want to add to the cluster as nodes, add the `node-network-cidr` key to the metadata (Edit VM -> Metadata) with a value equal to the cluster's `nodeNetworkCIDR`.

You can find out the `nodeNetworkCIDR` of the cluster using the command below:

```
d8 k get mc cloud-provider-yandex -o jsonpath='{.spec.settings.nodes.parameters.nodeNetworkCIDR}{"\n"}'
```

Creating a cluster in a new VPC and setting up a bastion host to access the nodes

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config.yml
   ```

2. Create a bastion host:

   ```
   yc compute instance create \
   --name bastion \
   --hostname bastion \
   --create-boot-disk image-family=ubuntu-2404-lts,image-folder-id=standard-images,size=20,type=network-hdd \
   --memory 2 \
   --cores 2 \
   --core-fraction 100 \
   --ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
   --zone ru-central1-a \
   --public-address 178.154.226.159
   ```

   > **i** Replace `<SSH_PUBLIC_KEY_FILE>` here with the name of your public key. For example, for a key with RSA encryption, it will be `id_rsa.pub`, and for a key with ED25519 encryption, it will be with `id_ed25519.pub`.

3. Continue installing the cluster by specifying the bastion host data. Answer `y` to the question about the Terraform cache:

   ```
   dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
   --ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --config=/config.yml
   ```

   > **i** Replace `<SSH_PRIVATE_KEY_FILE>` here with the name of your private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

Long switching to nodes in lower-priority groups

If switching to nodes in lower-priority groups takes a long time (for example, when node groups with preemptible instances are set to the highest priority and, if such instances are unavailable, switching to nodes from other groups takes a very long time), follow the instructions.

Replacing the OS image on the nodes

The `imageID` parameter of YandexInstanceClass defines the image for `CloudEphemeral` nodes. With a ModuleConfig-based provider configuration, this parameter also defines the image for `CloudPermanent` nodes and master nodes.

With a YandexClusterConfiguration-based configuration, the image for `CloudPermanent` nodes and master nodes is set by `imageID` in `nodeGroups[].instanceClass` and `masterNodeGroup.instanceClass`, respectively.

The replacement procedure is described in "How do I replace the OS image in a cloud cluster?".

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> **i** The module lifecycle stage: General Availability

Below is an example of the Yandex Cloud cloud provider configuration.

An example of the module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: b1g0ldjdpa8rtm2ru3s4
        folderID: b1gq1vn5jqk0tmh4u7bp
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 10.241.32.0/20
        sshPublicKey: ssh-rsa AAAAB3Nza...
        zones:
          - ru-central1-a
          - ru-central1-b
        labels:
          billing: prod
        externalSubnetIDs:
          worker:
            - e2lu8r1tbbtryhdpa9ro
        externalIPAddresses:
          worker:
            - Auto
    storage:
      parameters:
        excludedStorageClasses:
          - network-hdd
        provisionedStorageClasses:
          - name: network-ssd-64k
            type: network-ssd
            blockSize: 64Ki
    ccm:
      parameters:
        additionalExternalNetworkIDs:
          - enp6t4snovl2ko4p15em
        additionalInternalNetworkIDs:
          - enpmhcdmtvs6ptdq3mkt
```

The service account key is not stored in this resource, it is set in a separate Secret.

Using the ru-central1-f zone

To place nodes in `ru-central1-f`, explicitly list it in `zones` in place of one of the default zones. You can list up to four zones in `zones`. If the parameter is not set, Deckhouse Platform uses `ru-central1-a`, `ru-central1-b`, `ru-central1-d`, and `ru-central1-e`.

1. Select one of the default zones to replace with `ru-central1-f`. The following example excludes `ru-central1-e`.
2. List `ru-central1-f` and the other selected zones in the `zones` parameter of the `cloud-provider-yandex` ModuleConfig:

```
spec:
  settings:
    nodes:
      parameters:
        layout: Standard
        nodeNetworkCIDR: 10.241.32.0/20
        sshPublicKey: ssh-rsa AAAAB3Nza...
        zones:
          - ru-central1-a
          - ru-central1-b
          - ru-central1-d
          - ru-central1-f
```

Deckhouse Platform creates a subnet in `ru-central1-f` using the part of `nodeNetworkCIDR` assigned to the excluded zone. In a running cluster with the default zones, replacing `ru-central1-e` with `ru-central1-f` deletes the subnet in `ru-central1-e` and creates a new one in `ru-central1-f`. Subnets in the other selected zones are retained.

Using existing subnets

If `existingZoneToSubnetIDMap` is set, Deckhouse Platform uses the subnets specified by the parameter without creating or deleting them. The four-zone limit does not apply to `existingZoneToSubnetIDMap`. If `zones` is not set, nodes are placed in all zones specified in `existingZoneToSubnetIDMap`, including zones outside the default set.

Changing zones in a running cluster

To add `ru-central1-f`, update `zones`.

> **i** When you add `ru-central1-f`, Deckhouse Platform deletes subnets in the excluded default zones. Replacing `[ru-central1-a, ru-central1-b]` with `[ru-central1-a, ru-central1-b, ru-central1-f]` deletes the subnets in `ru-central1-d` and `ru-central1-e`. A subnet in `ru-central1-f` is created using the address range of one of the deleted subnets.

Changing the zone list may cause an address range conflict. This occurs if a subnet in a default zone you add needs to use the address range of the `ru-central1-f` subnet. Deckhouse Platform rejects such a change with an error. To resolve the conflict, apply the changes in two stages:

1. Remove `ru-central1-f` from `zones` and apply the changes. Deckhouse Platform deletes the subnet in `ru-central1-f`.
2. Set the final zone list including `ru-central1-f` and apply the changes again.

For example, this conflict may occur when replacing `ru-central1-d` with `ru-central1-e` while keeping `ru-central1-f` in the list.

If there is no conflict, you can change the zone list directly. The `ru-central1-f` subnet retains its address range.

If NAT subnets are not explicitly specified in the `WithNATInstance` layout, NAT instance placement is determined by `zones`:

Zone list	NAT instance placement
Contains only default zones	The NAT instance remains in `ru-central1-a`
Contains a zone outside the default set and excludes `ru-central1-a`	The NAT instance is recreated in the alphabetically first zone in `zones`

> **⚠** Cluster egress is interrupted while the NAT instance is recreated.

An example of the YandexInstanceClass custom resource

YandexInstanceClass describes the virtual machines that DP provisions for a node group. Only two parameters are required, `cores` and `memory`. The other parameters are optional.

The example below also sets the OS image, the disk type and size, the share of a vCPU, and the network interface mode:

```
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: worker
spec:
  cores: 4
  memory: 8192
  coreFraction: 50
  imageID: fd8bhu2b2bsbnqkhhd2k
  diskType: network-ssd
  diskSizeGB: 50
  networkType: SoftwareAccelerated
```

LoadBalancer

Service annotations

Default values are configured in the cluster for placing load balancer resources (the network for the target group and the subnet for the Listener). These values are set automatically during cluster setup and can be overridden with annotations at the individual Service level.

> **i** The default behavior (external or internal LB) depends on the cluster configuration. To explicitly choose the type, use the `yandex.cpi.flant.com/loadbalancer-external` annotation.

The following annotations are supported by Yandex Cloud Controller Manager:

1. `yandex.cpi.flant.com/target-group-network-id` — specifies the NetworkID in which the target group for this Service will be created. Overrides the corresponding default value.
2. `yandex.cpi.flant.com/listener-subnet-id` — sets the SubnetID for the Listeners of the LB created for this Service.

Overrides the corresponding default value.
3. `yandex.cpi.flant.com/listener-address-ipv4` — sets a predefined IPv4 address for the Listeners (supported for both internal and external LBs).
4. `yandex.cpi.flant.com/loadbalancer-external` — enables creation of an external LB for this Service (use it when you need to explicitly create an external load balancer). Overrides the default behavior.
5. `yandex.cpi.flant.com/target-group-name-prefix` — specifies the name prefix of the target group used by the LoadBalancer. The annotation on the Service identifies the target group but does not select nodes. To include nodes in the target group, set the same annotation value in `spec.nodeTemplate.annotations` in the NodeGroup. Yandex Cloud Controller Manager includes all suitable nodes carrying this annotation value, regardless of their NodeGroup. The target group name is formed as `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`. For details, see “Using a separate target group for a NodeGroup”.

If separate target groups are created for the control plane or master nodes, add the label `node.kubernetes.io/exclude-from-external-load-balancers: ""` to the master nodes. This prevents the controller from automatically adding master nodes to new target groups for load balancers.

If you create your own load balancer for master nodes and want YCC to also be able to place its load balancers on master nodes, pre-create a target group with a name matching the pattern `${CLUSTER-NAME}${VPC.ID}`.

Target group health checks

Health check parameters (for LB target groups created by the controller):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — how often to run the check, in seconds (default: 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — how long to wait for an endpoint response, in seconds. If no response is received within this time, the check is considered failed (default: 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — how many consecutive failed checks are required to mark an endpoint as unhealthy and exclude it from load balancing (default: 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — how many consecutive successful checks are required to return an endpoint to healthy status and include it back in load balancing (default: 2).

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Layouts

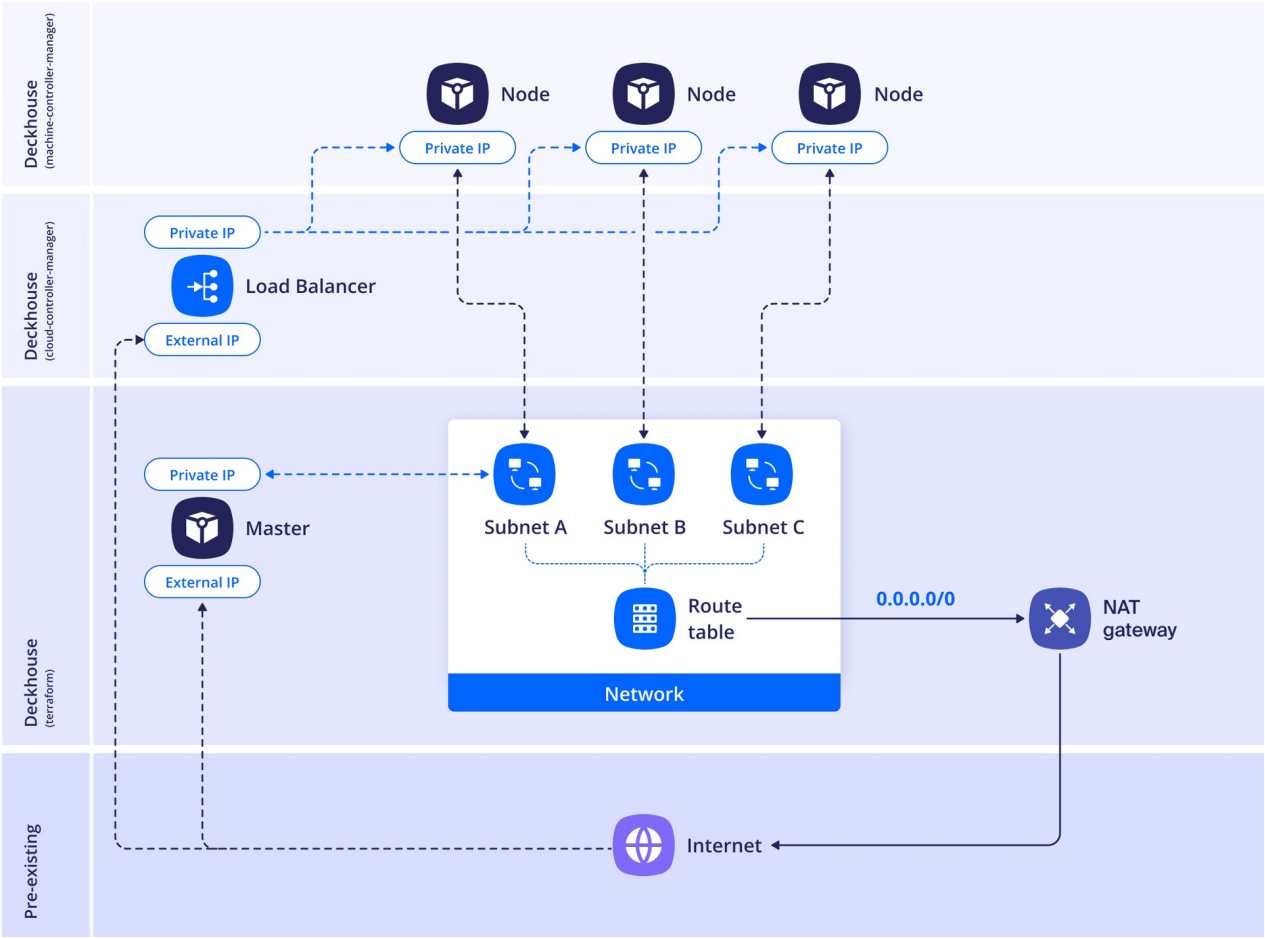
> The module lifecycle stage: General Availability

Three layouts are supported. The layout is specified by the `layout` parameter in the section of the `cloud-provider-yandex` module’s ModuleConfig resource. Below is more information about each of them.

Standard

> In this placement strategy, nodes do not have public IP addresses allocated to them; they use NAT gateway service in Yandex Cloud to connect to the Internet. NAT Gateway uses random public IP addresses from dedicated ranges. Because of this, it is impossible to whitelist the IP addresses of cloud resources located behind a specific NAT gateway on the side of other services.

> Because nodes are created without public IP addresses in this layout, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly over a private network or through a bastion host. If the master node is not directly accessible, run the installation with the `--ssh-bastion-host`, `--ssh-bastion-user`, and, if needed, `--ssh-bastion-port` parameters.



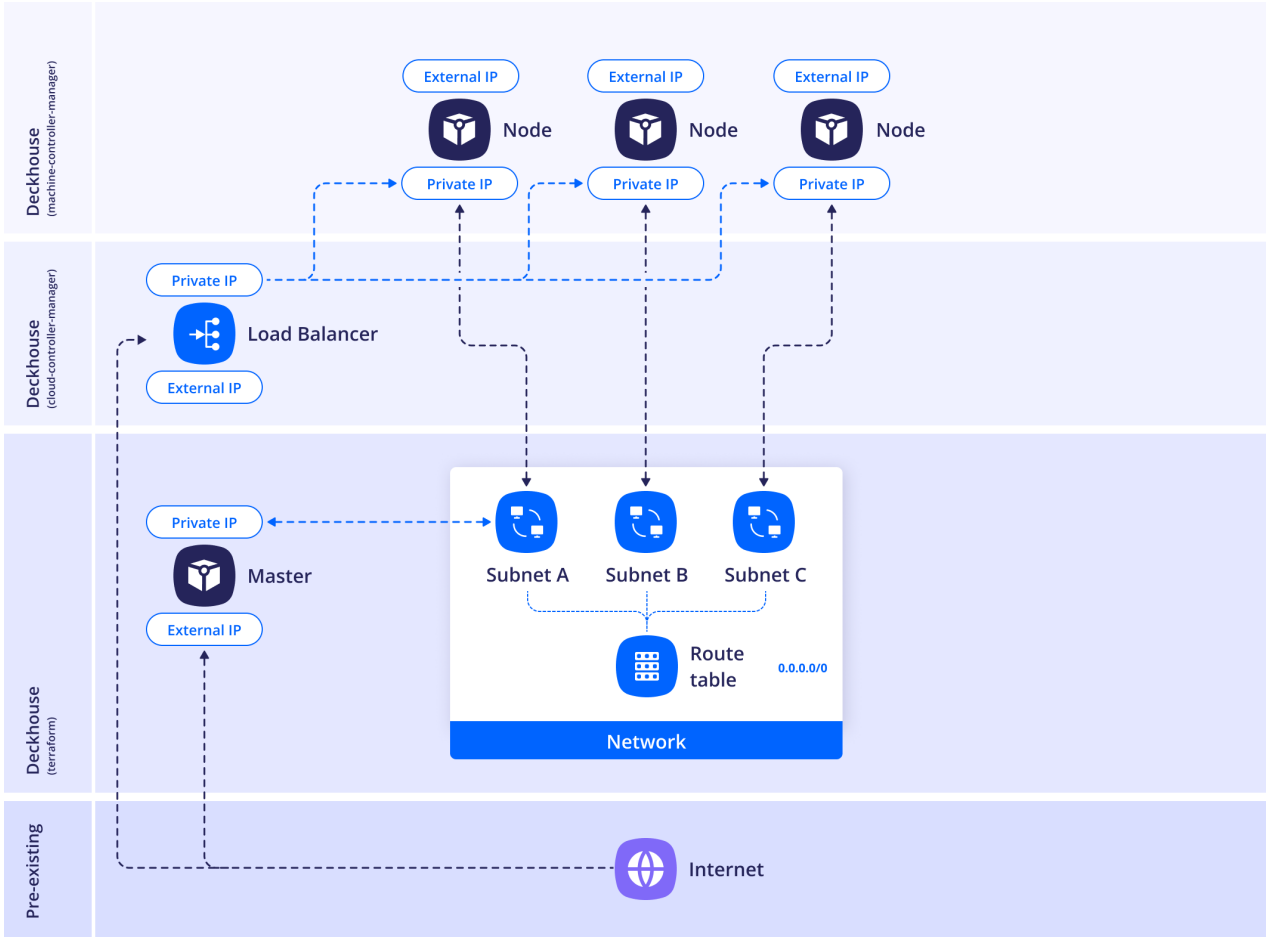
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
          - ru-central1-b
          - ru-central1-d
        labels:
          billing: prod
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

WithoutNAT

In this layout, NAT (of any kind) is not used, and each node is assigned a public IP.

⚠ The `cloud-provider-yandex` module does not assign security groups to the nodes, so in this layout the nodes are accessible from the outside without restrictions. To restrict the traffic, use the rules of the default group as described in “Configuring security groups”.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: WithoutNAT
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
          - ru-central1-b
          - ru-central1-d
        externalIPAddresses:
          master:
            - Auto
            - Auto
            - Auto
          worker:
            - <ZONE_A_EXTERNAL_IP_WORKER_1>
            - Auto
        externalSubnetIDs:
          master:
            - <ZONE_A_SUBNET_ID>
            - <ZONE_B_SUBNET_ID>
            - <ZONE_D_SUBNET_ID>
          worker:
            - <ZONE_A_SUBNET_ID>
            - <ZONE_B_SUBNET_ID>
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

WithNATInstance

In this layout, a NAT instance is created in a separate subnet, and the route table of the zone subnets gets a `0.0.0.0/0` route with the NAT instance as the next hop. The subnet is allocated to prevent routing loops and must not overlap with other networks used in the cluster.

To place the NAT instance in an existing subnet, use the `withNATInstance.internalSubnetID` parameter — the instance will be created in the zone corresponding to that subnet.

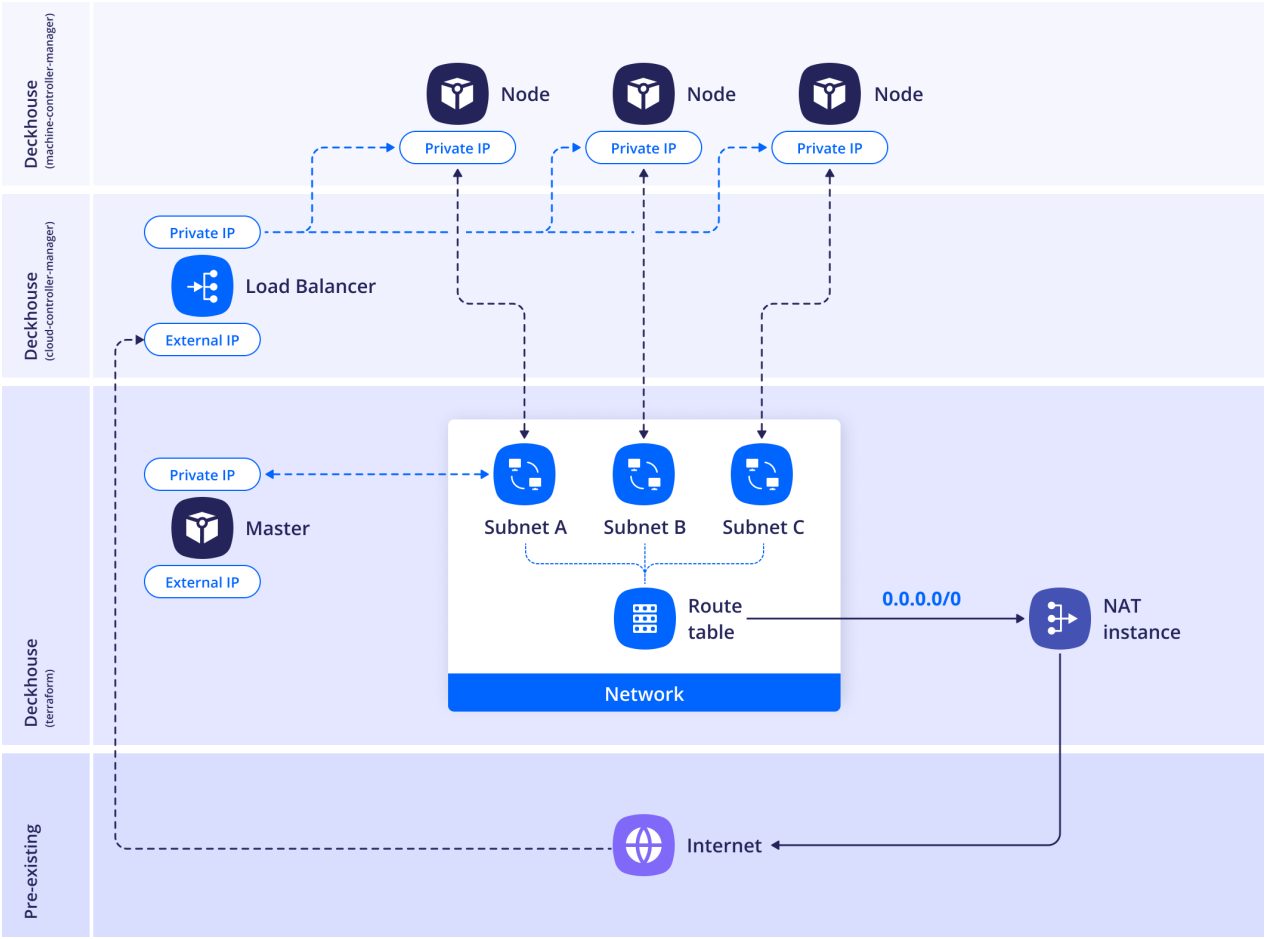
If you need to create a new subnet, specify the `withNATInstance.internalSubnetCIDR` parameter — the NAT instance will be deployed in it.

One of the parameters — `withNATInstance.internalSubnetID` or `withNATInstance.internalSubnetCIDR` — is required.

When the `withNATInstance.internalSubnetCIDR` parameter is specified, the NAT instance will be created in a single instance in zone `a`.

When the `withNATInstance.internalSubnetID` parameter is specified, the NAT instance will be created in a single instance in the zone specified in the designated network settings.

If the `withNATInstance.externalSubnetID` is provided in addition to previous ones, the NAT instance will be attached to it via secondary interface.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        cloudID: <CLOUD_ID>
        folderID: <FOLDER_ID>
    nodes:
      parameters:
        layout: WithNATInstance
        sshPublicKey: <SSH_PUBLIC_KEY>
        nodeNetworkCIDR: 192.168.12.0/24
        existingNetworkID: <EXISTING_NETWORK_ID>
        zones:
          - ru-central1-a
        withNATInstance:
          natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
          internalSubnetID: <INTERNAL_SUBNET_ID>
          externalSubnetID: <EXTERNAL_SUBNET_ID>
        dhcpOptions:
          domainName: test.local
          domainNameServers:
            - <DNS_SERVER_1>
            - <DNS_SERVER_2>
```

> **i** In this example, master and worker nodes are created without public IP addresses and without additional external network interfaces. Outbound traffic from the cluster subnets goes through the NAT instance.
>
> The NAT instance is not automatically used by the `dhctl` installer as a bastion host or jump host. To install the cluster, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly or through a bastion host.
>
> If the master node does not have a public IP address or is not directly accessible, run the installation with the `--ssh-bastion-host` and `--ssh-bastion-user` parameters.
>
> If you need to connect directly to a master node using a public IP address, specify the `externalIPAddresses` and `externalSubnetIDs` parameters in the `nodes.parameters` section of the `cloud-provider-yandex` module's ModuleConfig resource. Note that the interface connected to the network specified in `externalSubnetIDs` will be used as the node's default gateway.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Cloud provider — zVirt

> **i** Included in extensions: **On-prem IaaS**

> **i** The module lifecycle stage: General Availability

The `cloud-provider-zvirt` module integrates Deckhouse Platform with zVirt. It allows the `node-manager` module to use zVirt resources when provisioning nodes for a NodeGroup.

The module is automatically enabled for all cloud clusters deployed in zVirt.

Features of the `cloud-provider-zvirt` module:

- Managing zVirt resources via `cloud-controller-manager`, which updates Kubernetes node addresses and removes from Kubernetes the nodes whose virtual machines no longer exist in zVirt.
- Provisioning disks via the zVirt CSI driver (`csi.ovirt.org`) and creating StorageClasses for zVirt storage domains so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-ovirt`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the ZvirtInstanceClass resource.
- Registering with `node-manager` so that ZvirtInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> **⚠** If the cluster was installed with the ZvirtClusterConfiguration schema, it must be migrated to the ModuleConfig-based configuration. Until the migration is complete, the `D8CloudProviderZvirtMigrationPending` alert may fire and Deckhouse Platform updates may be blocked.
>
> The migration procedure is described in "How to migrate a cloud provider to ModuleConfig-based configuration?".

Third-party components

List of third-party software used in the `cloud-provider-zvirt` module:

- zVirt CSI driver release-4.22

  License: Apache License 2.0

  CSI driver for zVirt (OpenShift oVirt CSI driver).

- zVirt Terraform provider 2.2.0

  License: BSD 3-Clause License

  Terraform provider for zVirt (oVirt).

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** Included in extensions: **On-prem IaaS**

> The module lifecycle stage: General Availability

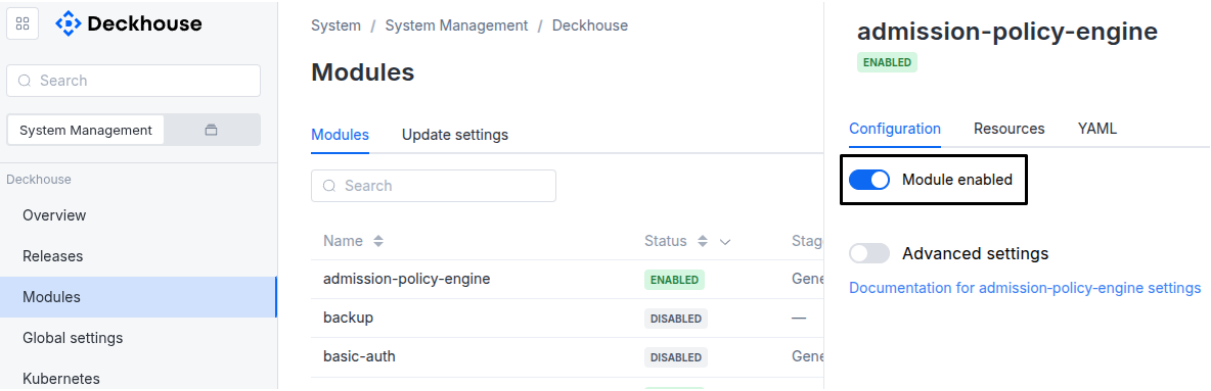
The module has 1 alert.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-zvirt` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-zvirt
```

- **Using ModuleConfig `cloud-provider-zvirt`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-zvirt` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-zvirt`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  enabled: true
```

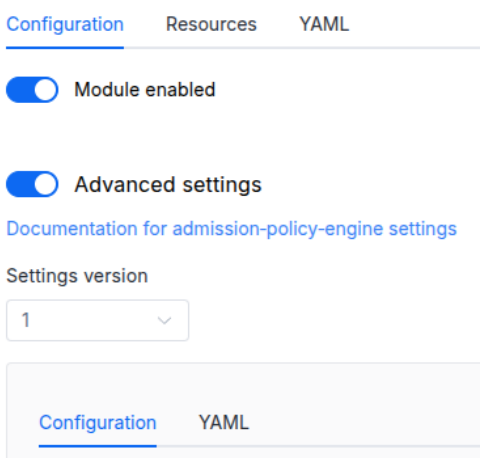
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-zvirt` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cloud-provider-zvirt` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-zvirt`) by

specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-zvirt` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-zvirt
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-zvirt` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-zvirt`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-zvirt
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  The settings are grouped under the `provider`, `nodes`, `storage`, and `ccm` parameters.

  The conversion sets temporary values for the `server`, `clusterID`, and `sshPublicKey` parameters. Replace them with the actual values.

  The zVirt login and password are not stored in the module settings. The module reads them from the `d8-credentials` Secret of the `cloud-provider.deckhouse.io/credentials` type.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.ccm**
    object

    Cloud Controller Manager (CCM) subsystem settings.

    CCM keeps the cluster node metadata up to date and removes from the cluster the nodes whose virtual machines no longer exist in zVirt.

    - **settings.ccm.disabled**
      boolean

      Disables the Cloud Controller Manager.

      When set to `true`, the module does not deploy CCM.

      Default: `false`

    - **settings.ccm.parameters**
      object

      Required value

      Parameters of the CCM subsystem.

  - **settings.nodes**
    object

    Required value

    Nodes subsystem settings.

    The subsystem describes how the cluster nodes are created in zVirt, including the layout, the SSH key, and the static network configuration.

    - **settings.nodes.disabled**

boolean

Disables the nodes subsystem.

When set to `true`, the cluster does not create CloudEphemeral nodes in zVirt.

Default: `false`

- **settings.nodes.parameters**
  object

  Required value

  Parameters of the nodes subsystem.

  - **settings.nodes.parameters.customNetworkConfigs**
    object

    Static network configuration of the nodes of CloudPermanent NodeGroups.

    The key is the name of a CloudPermanent NodeGroup, for example `master`. Nodes of a NodeGroup that is not listed get their network settings over DHCP.

    Addresses from the `networkInterfaceAddresses` list are assigned by node index. The first address goes to the node with index 0, the second address goes to the node with index 1, and so on. If the list holds fewer addresses than the NodeGroup has replicas, the remaining nodes get their addresses over DHCP.

    - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>**
      object

      `<KEY_NAME>` — item (key) name.

      Static network configuration of a node network interface.

      - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>.dnsServers**
        array of strings

        List of DNS servers.

        Example:

        ```
        dnsServers:
        - 8.8.8.8
        - 8.8.4.4
        ```

      - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>.networkInterfaceAddresses**
        array of strings

        Required value

        List of IP addresses to assign to the interface, one address per node.

        Example:

        ```
        networkInterfaceAddresses:
        - 192.168.1.10
        - 192.168.1.11
        ```

      - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>.networkInterfaceGateway**
        string

        Required value

        Default network gateway.

        Example:

        ```
        networkInterfaceGateway: 192.168.1.1
        ```

      - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>.networkInterfaceName**
        string

        Required value

        Name of the network interface to apply the static configuration to.

        Example:

        ```
        networkInterfaceName: enp1s0
        ```

      - **settings.nodes.parameters.customNetworkConfigs.<KEY_NAME>.networkInterfaceNetmask**
        string

        Required value

        Subnet mask for the interface.

        Example:

```
networkInterfaceNetmask: 255.255.255.0
```

- **settings.nodes.parameters.layout**
  string

  Required value

  Layout name.

  The available layouts are described in “Layouts”.

  Allowed values: `Standard`

- **settings.nodes.parameters.sshPublicKey**
  string

  Required value

  Public SSH key for accessing the nodes.

- **settings.provider**
  object

  Required value

  zVirt connection settings.

  - **settings.provider.parameters**
    object

    Required value

    Parameters for connecting to the zVirt API.

    The login and the password are not set in the module settings. The module reads them from the `d8-credentials` Secret of the `cloud-provider.deckhouse.io/credentials` type in the `d8-cloud-provider-zvirt` namespace.

    - **settings.provider.parameters.caBundle**
      string

      Base64-encoded bundle of CA certificates in PEM format to verify the zVirt API certificate.

      When the `insecure` parameter is set to `true`, the bundle is ignored.

    - **settings.provider.parameters.clusterID**
      string

      Required value

      ID of the zVirt cluster to create virtual machines in.

      The cluster must have shared storage domains and CPUs of the same type.

      Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

      Example:

      ```
      clusterID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
      ```

    - **settings.provider.parameters.insecure**
      boolean

      Disables verification of the zVirt API certificate.

      Set to `true` if zVirt uses a self-signed certificate.

      Default: `false`

    - **settings.provider.parameters.server**
      string

      Required value

      URL of the zVirt API endpoint.

      Specify a URL with the `http` or `https` scheme.

- **settings.storage**
  object

  Storage subsystem settings.

  The subsystem provides the CSI driver and the StorageClasses for zVirt storage domains.

  - **settings.storage.disabled**
    boolean

    Disables the storage subsystem.

    When set to `true`, disk provisioning in the cluster is unavailable.

    Default: `false`

  - **settings.storage.parameters**
    object

Required value

Parameters of the storage subsystem.

- **settings.storage.parameters.excludedStorageClasses**
  array of strings

  List of StorageClasses to exclude from creation in the cluster.

  The module creates a StorageClass for each active zVirt storage domain. Each item of the list is a regular expression matched against the whole StorageClass name. For example, `data` excludes only the `data` StorageClass, and `data.*` excludes every StorageClass whose name starts with `data`.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Provider configuration

The module lifecycle stage: General Availability

ZvirtClusterConfiguration

Version: deckhouse.io/v1

Warning. The resource is deprecated. The cloud provider configuration is set in the ModuleConfig resource of the `cloud-provider-zvirt` module. Migrate the cluster to ModuleConfig as described in “How to migrate a cloud provider to ModuleConfig-based configuration?”.

Describes the configuration of a cloud cluster in zVirt.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

```
d8 system edit provider-cluster-configuration
```

After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
clusterID: b46372e7-0d52-40c7-9bbf-fda31e187088
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
    storageDomainID: c4bf82a5-b803-40c3-9f6c-b9398378f424
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **clusterID**
  string

  Required value

  Cluster ID with shared storage domains and CPUs of the same type to create virtual machines.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  clusterID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **kind**
  string

  Required value

  Allowed values: `ZvirtClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the ZvirtInstanceClass.

    - **masterNodeGroup.instanceClass.customNetworkConfig**
      object

      Enables a custom static network configuration for nodes in this NodeGroup.

      If specified, you must set `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` and `networkInterfaceNetmask`.

      - **masterNodeGroup.instanceClass.customNetworkConfig.dnsServers**
        string

        List of DNS servers.

        Example:

        ```
        dnsServers: 8.8.8.8 8.8.4.4
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceAddress**
        array of strings

        List of IP addresses to assign to the interface (one IP address per node).

        Example:

        ```
        networkInterfaceAddress:
        - 192.168.1.10
        - 192.168.1.11
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceGateway**
        string

        Default network gateway.

        Example:

        ```
        networkInterfaceGateway: 192.168.1.1
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceName**
        string

        Name of the network interface to apply the static configuration to.

        Example:

```
networkInterfaceName: enp1s0
```

- **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceNetmask**
  string

  Subnet mask for the interface.

  Example:

  ```
  networkInterfaceNetmask: 255.255.255.0
  ```

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in GiB.

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **masterNodeGroup.instanceClass.numCPUs**
  integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **masterNodeGroup.instanceClass.storageDomainID**
  string

  Required value

  Storage domain id which contains the shared resources that must be available to all datacenter hosts.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **masterNodeGroup.instanceClass.template**
  string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **masterNodeGroup.instanceClass.vnicProfileID**
  string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

Example:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the.

    - **nodeGroups.instanceClass.customNetworkConfig**
      object

      Enables a custom static network configuration for nodes in this NodeGroup.

      If specified, you must set `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` and `networkInterfaceNetmask`.

      - **nodeGroups.instanceClass.customNetworkConfig.dnsServers**
        string

        List of DNS servers.

        Example:

        ```
        dnsServers: 8.8.8.8 8.8.4.4
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceAddress**
        array of strings

        List of IP addresses to assign to the interface (one IP address per node).

        Example:

        ```
        networkInterfaceAddress:
        - 192.168.1.10
        - 192.168.1.11
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceGateway**
        string

        Default network gateway.

        Example:

        ```
        networkInterfaceGateway: 192.168.1.1
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceName**
        string

        Name of the network interface to apply the static configuration to.

        Example:

        ```
        networkInterfaceName: enp1s0
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceNetmask**
        string

        Subnet mask for the interface.

        Example:

        ```
        networkInterfaceNetmask: 255.255.255.0
        ```

    - **nodeGroups.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in GiB.

      Default: `10`

      Example:

```
etcdDiskSizeGb: 10
```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.numCPUs**
  integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **nodeGroups.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **nodeGroups.instanceClass.storageDomainID**
  string

  Storage domain id which contains the shared resources that must be available to all datacenter hosts.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **nodeGroups.instanceClass.template**
  string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **nodeGroups.instanceClass.vnicProfileID**
  string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**
  object

  A list of labels to attach to cluster resources.

  The same as the `metadata.labels` standard field.

  Example:

  ```
  labels:
    environment: production
    app: warp-drive-ai
  ```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **provider**
  object

  Required value

  Parameters for connecting to the Zvirt.

  - **provider.caBundle**
    string

    CA certificate in base64.

    Default:

  - **provider.insecure**
    boolean

    Set to `true` if Zvirt has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    Required value

    The user’s password.

  - **provider.server**
    string

    Required value

    The URL to the Zvirt API endpoint.

  - **provider.username**
    string

    Required value

    The login ID.

- **sshPublicKey**
  string

  Required value

  a public key for accessing nodes.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

> The module lifecycle stage: General Availability

Environment requirements

The provider requires the following:

- A working zVirt installation with an accessible API
- zVirt version `4.0–4.4`
- A user with the permissions listed in the “User permissions” section
- A storage domain with an uploaded cloud image of the operating system
- A virtual machine template prepared from the cloud image

If the zVirt API uses a self-signed certificate, pass the CA certificate in the `caBundle` parameter or set the `insecure` parameter to `true`.

User permissions

Deckhouse Platform does not require the `SuperUser` role for the zVirt user. It is enough to assign the following predefined zVirt roles to the user on the objects the cluster works with:

Role	Object	Purpose
`UserVmManager`	zVirt cluster from the `clusterID` parameter	Creating, starting, stopping, and deleting node virtual machines, creating their network interfaces, attaching and detaching disks
`UserTemplateBasedVm`	Virtual machine template	Creating virtual machines from the template
`DiskOperator`	Storage domains that contain the template disk and where disks for etcd and cluster volumes are created	Creating, resizing, attaching, detaching, and deleting disks
`VnicProfileUser`	vNIC profile	Creating a virtual machine network interface from the vNIC profile
`TagManager`	Entire system (system permission)	Creating the cluster tag and assigning it to virtual machines of CloudEphemeral nodes

The `TagManager` role is assigned only on the entire system, because zVirt checks the tag management permission at this level.

> ⚠ The `TagManager` role is administrative. With it, the user can view all objects via the zVirt API but can modify only the objects on which the roles from the table are assigned to the user.

zVirt assigns some roles to the Everyone group by default, so you do not need to assign them to the user separately:

- `UserTemplateBasedVm` on a template created with the “Allow all users to access this Template” checkbox selected
- `VnicProfileUser` on a public vNIC profile
- `CpuProfileOperator` on the cluster CPU profile
- `DiskProfileUser` on the disk profiles of storage domains

If these permissions were removed in your zVirt installation, assign the roles to the user.

To assign a role on an object:

1. Open the object in the zVirt Administration Portal, for example, a cluster in the “Compute → Clusters” section.
2. Go to the “Permissions” tab and click “Add”.
3. Select the user and the role and save the permission.

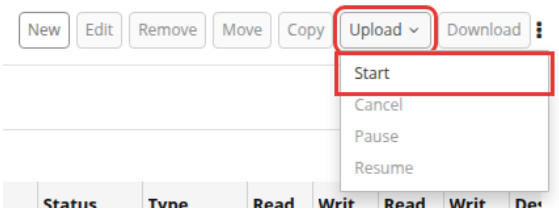
Assign the `TagManager` role in the “Administration → Configure” section on the “System Permissions” tab.

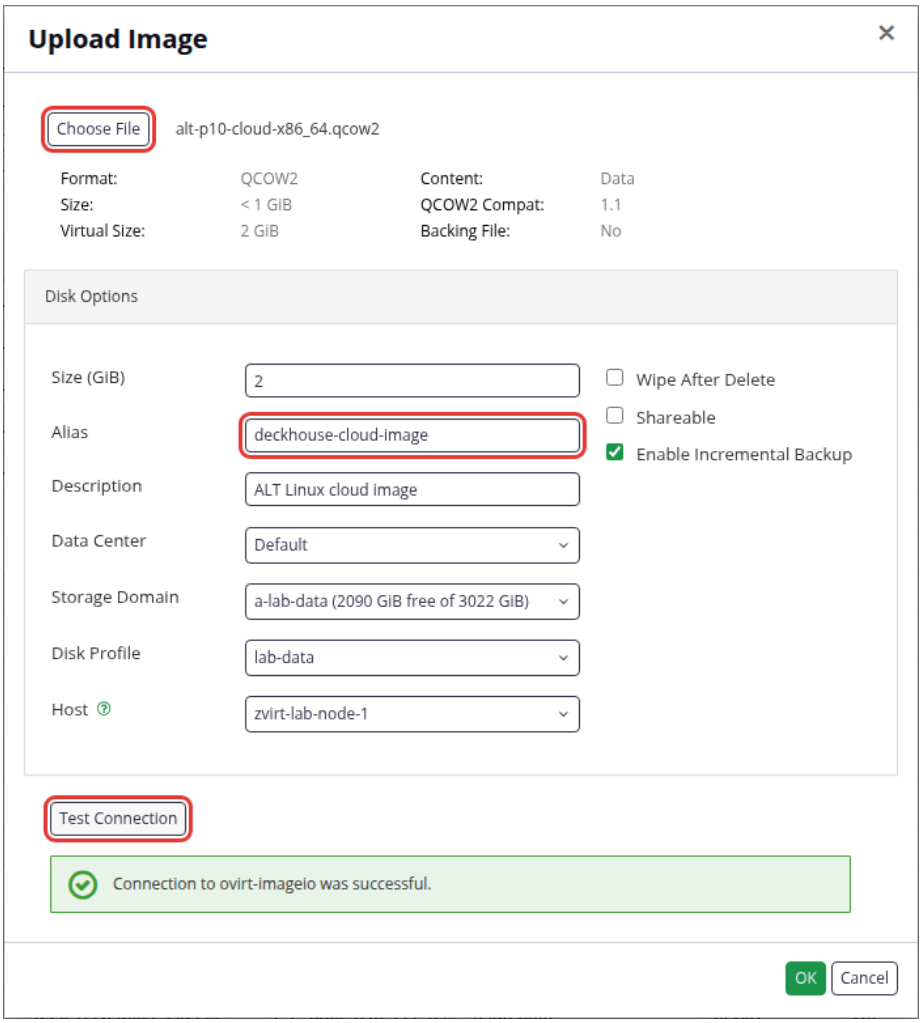
Prepare an operating system image

Operating system (OS) vendors usually provide special cloud builds of their OSes for virtualization environments. These builds contain virtual hardware drivers, `cloud-init`, and virtualization guest agents and are distributed as IMG or QCOW2 disk images. Use these cloud images as the OS of the cluster nodes.

The OS cloud image must be placed in the zVirt disk storage. To upload the image to the storage:

1. Open the zVirt Administration Portal and go to the “Storage → Disks” section.
2. Upload the OS cloud image to the storage.





3. Wait until the image upload is complete. The `OK` status must appear in the “Status” column.

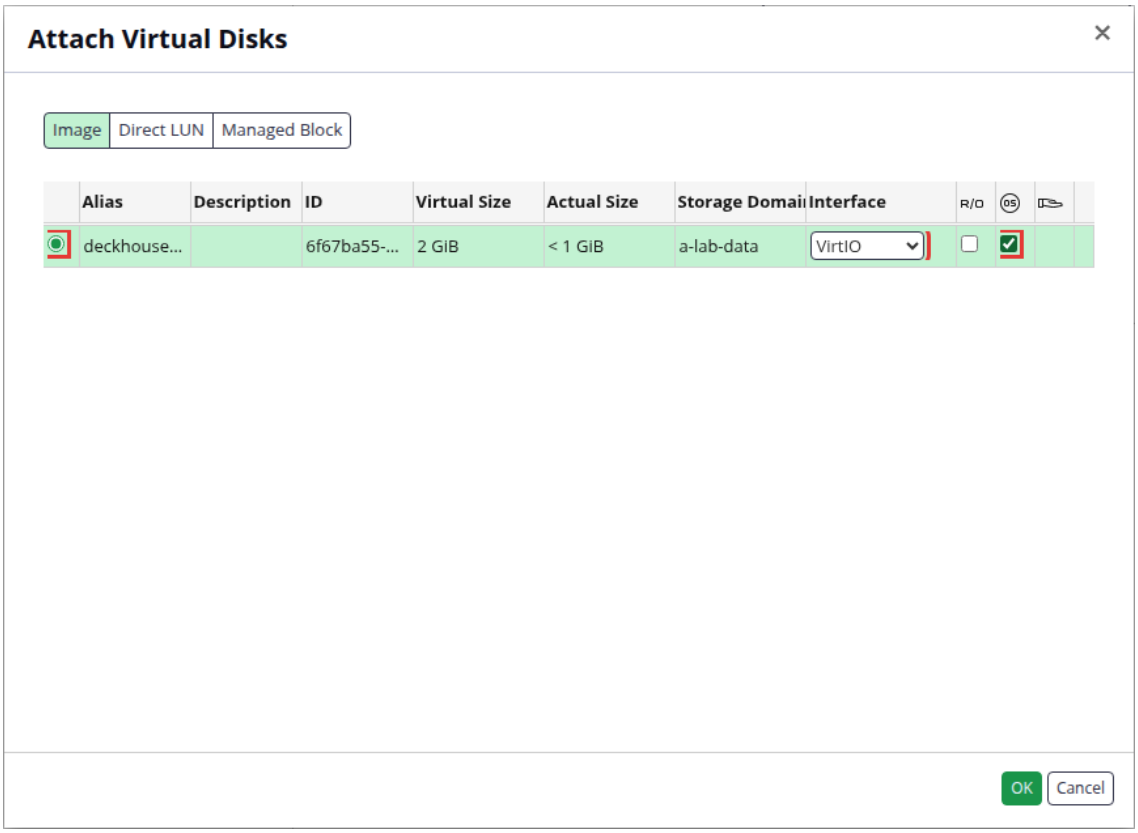


Prepare a virtual machine template

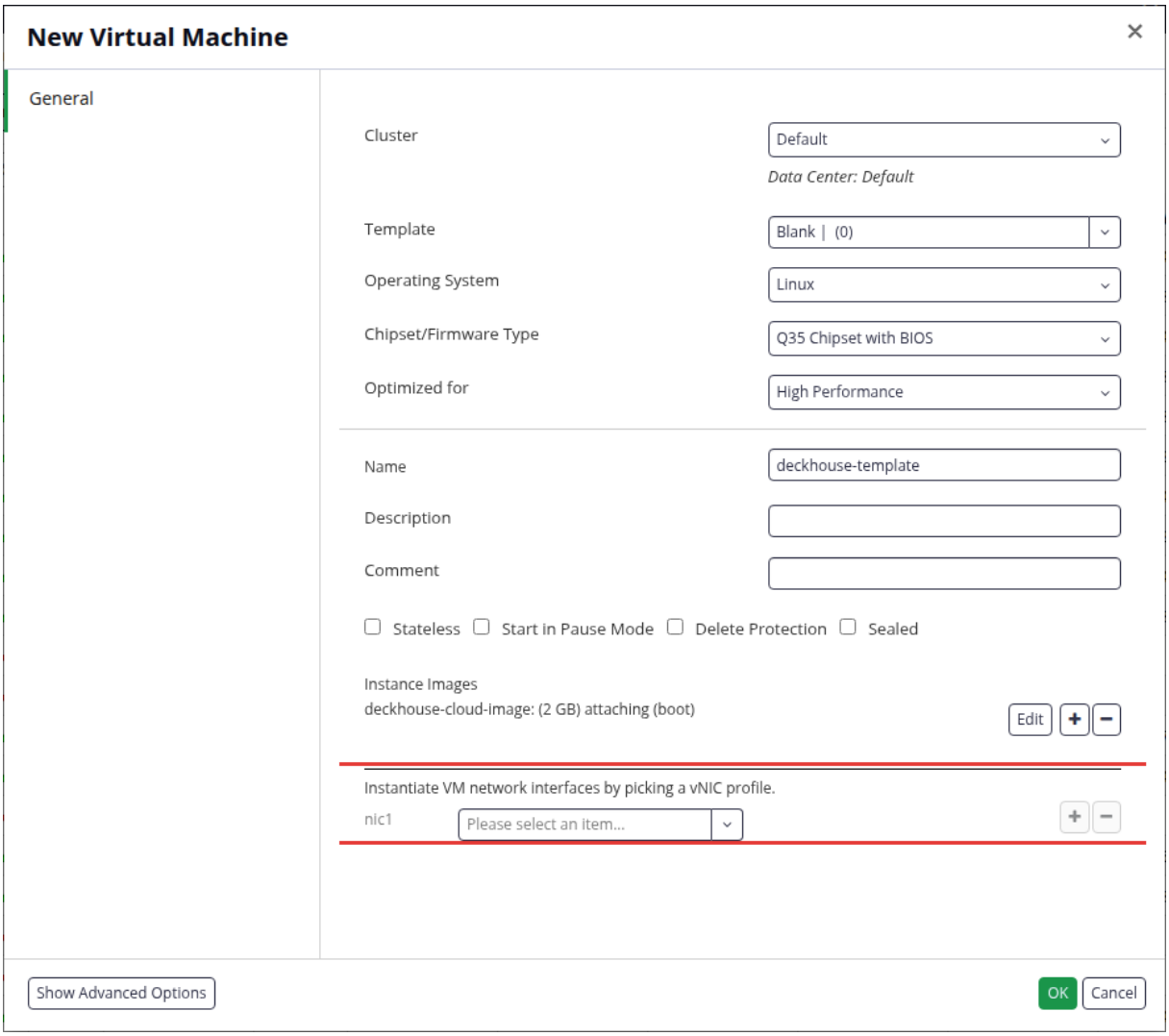
> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

The template is created from a virtual machine (VM) with the OS cloud image attached. To prepare the template:

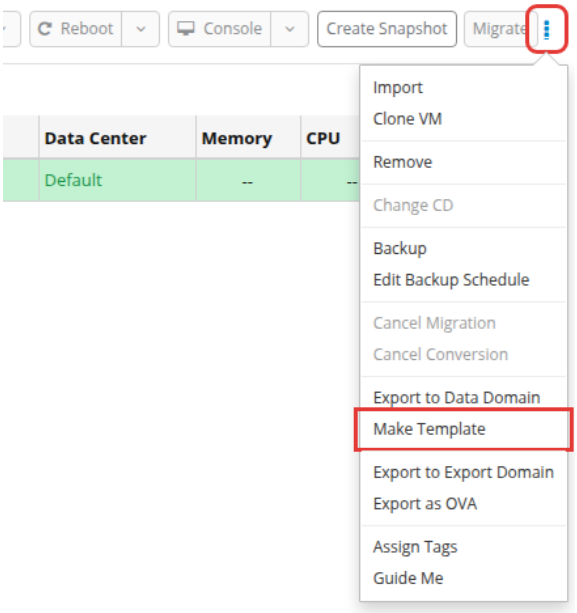
1. In the zVirt Administration Portal, go to the “Compute → Virtual Machines” section and create a VM. In the “General” section, set the following parameters:
   - “Template”: `Blank`.
   - “Operating System”: The OS of the cloud image.
   - “Optimized for”: “High Performance”.
   - “Instance Images”: Click “Attach”, select the uploaded OS cloud image, select the checkbox in the “OS” column to make the disk bootable, and select the “VirtIO” interface. Do not create or attach other disks.



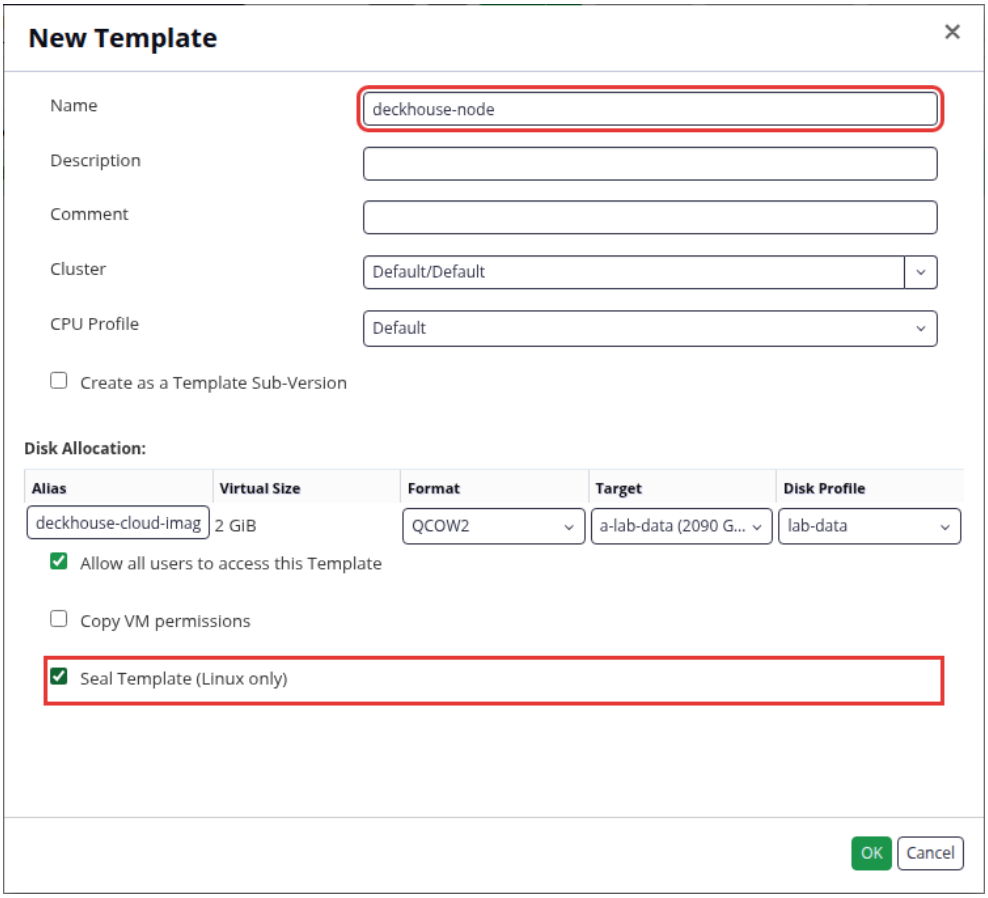
2. Do not attach network interfaces. Deckhouse Platform creates a network interface when it creates a VM for a node.



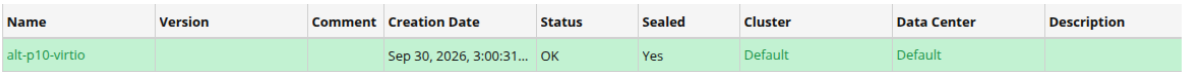
3. Leave the other parameters at their defaults and create the VM.
4. When the VM is created, select it in the list, open the additional actions menu (⋮), and select “Make Template”.



5. Specify the template name and enable the “Seal Template (Linux only)” parameter. The other template parameters are optional.



6. Wait until the template is created. While the template is being created, the VM and the cloud image disk are in the “Image Locked” state. The created template is displayed in the “Compute → Templates” section.



Replacing the OS image

The `template` parameter of ZvirtInstanceClass defines the image for `CloudEphemeral` nodes. With a ModuleConfig-based provider configuration, this parameter also defines the image for `CloudPermanent` nodes and master nodes.

With a ZvirtClusterConfiguration-based configuration, the image for `CloudPermanent` nodes and master nodes is set by `template` in `nodeGroups[].instanceClass` and `masterNodeGroup.instanceClass`, respectively.

The replacement procedure is described in “How do I replace the OS image in a cloud cluster?” .

Credentials Secret

The zVirt user login and password are stored in a separate Secret rather than in the ModuleConfig. The module reads it and passes the credentials to its components that access the zVirt API.

The Secret is created during cluster installation together with the other resources of the initial configuration. The Secret must meet the following requirements:

- The name is `d8-credentials` and the namespace is `d8-cloud-provider-zvirt`.
- The type is `cloud-provider.deckhouse.io/credentials`.
- The `authScheme` field is set to `userPassword`.
- The `identity` field contains the zVirt user login.
- The `secret` field contains the zVirt user password.

```
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-zvirt
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: userPassword
  identity: <USERNAME>
  secret: <PASSWORD>
```

Replace `<USERNAME>` and `<PASSWORD>` with the zVirt user login and password.

A Secret that does not meet these requirements fails the check both during cluster installation and when applied in a running cluster.

To change the credentials, update the `identity` and `secret` fields:

```
d8 k -n d8-cloud-provider-zvirt edit secret d8-credentials
```

In clusters migrated to ModuleConfig, the login and password are moved to this Secret from the `provider.username` and `provider.password` parameters of the ZvirtClusterConfiguration resource.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

ZvirtInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of the zVirt virtual machines that Deckhouse Platform creates for the cluster nodes.

The resource is referenced in the `cloudInstances.classReference` parameter of a NodeGroup.

- **metadata**
  object
- **spec**
  object

  Required value

  Defines the parameters of a zVirt virtual machine.

  - **spec.etcdDiskSizeGb**
    integer

    Size of the etcd disk, in GiB.

    The parameter is required in the instance class of the `master` NodeGroup and cannot be set in the instance classes of other NodeGroups.

    Allowed values: `1 <= X`

    Example:

    ```
    etcdDiskSizeGb: 10
    ```

- **spec.memory**
  integer

  Required value

  Amount of memory allocated to the virtual machine, in MiB.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **spec.numCPUs**
  integer

  Required value

  Number of vCPUs allocated to the virtual machine.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **spec.rootDiskSizeGb**
  integer

  Size of the virtual machine root disk, in GiB.

  The root disk created from the template is resized to this value.

  If the parameter is not set, the disk size is 50 GiB for CloudPermanent nodes and 20 GiB for CloudEphemeral nodes.

  Allowed values: `1 <= X`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **spec.storageDomainID**
  string

  ID of the zVirt storage domain to create the etcd disk of master nodes in.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **spec.template**
  string

  Required value

  Name of the zVirt template to clone virtual machines from.

  Example:

  ```
  template: debian-bookworm
  ```

- **spec.vnicProfileID**
  string

  Required value

  ID of the virtual network interface card (vNIC) profile used to create the virtual machine network interface.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **status**
  object

  Status of the ZvirtInstanceClass resource.

  - **status.nodeGroupConsumers**
    array of strings

    Names of the NodeGroups that use this instance class.

Parameters of the zVirt virtual machines that Deckhouse Platform creates for the cluster nodes.

The resource is referenced in the `cloudInstances.classReference` parameter of a NodeGroup.

- **metadata**

object

- **spec**
  object

  Required value

  - **spec.memory**
    integer

    Required value

    Amount of memory allocated to the virtual machine, in MiB.

    Example:

    ```
    memory: 8192
    ```

  - **spec.numCPUs**
    integer

    Required value

    Number of vCPUs allocated to the virtual machine.

    Example:

    ```
    numCPUs: 2
    ```

  - **spec.rootDiskSizeGb**
    integer

    Size of the virtual machine root disk, in GiB.

    The root disk created from the template is resized to this value.

    Default: `30`

    Example:

    ```
    rootDiskSizeGb: 30
    ```

  - **spec.template**
    string

    Required value

    Name of the zVirt template to clone virtual machines from.

    Example:

    ```
    template: debian-bookworm
    ```

  - **spec.vnicProfileID**
    string

    Required value

    ID of the virtual network interface card (vNIC) profile used to create the virtual machine network interface.

    Pattern: `^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}$`

    Example:

    ```
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
    ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

    Names of the NodeGroups that use this instance class.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> The module lifecycle stage: General Availability

Provider configuration

All provider settings except the credentials are set in the ModuleConfig resource of the module. Deckhouse Platform applies them both when creating the cluster and in a running cluster.

The login and password for accessing the zVirt API are not stored in the module settings. The module reads them from the `d8-credentials` Secret, whose format is covered in the “Credentials Secret” section.

Virtual machine parameters are set in the ZvirtInstanceClass resource referenced by a NodeGroup. An example configuration with all the resources is given on the “Layouts” page.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> **i** The module lifecycle stage: General Availability

How to get the vNIC profile ID?

You can get the ID of the vNIC profile set in the `vnicProfileID` parameter of the ZvirtInstanceClass resource by querying the zVirt API:

1. Get an access token for the zVirt API:

   ```
   curl -s -X POST <ZVIRT_ENGINE_URL>/sso/oauth/token \
     -H "Accept: application/json" \
     --data-urlencode "grant_type=password" \
     --data-urlencode "scope=ovirt-app-api" \
     --data-urlencode "username=<USERNAME>" \
     --data-urlencode "password=<PASSWORD>"
   ```

   Replace `<ZVIRT ENGINE URL>` with the URL from the `server` parameter of the module settings without the trailing `/api`, for example, `https://zvirt.example.com/ovirt-engine`. Replace `<USERNAME>` and `<PASSWORD>` with the zVirt user login and password in the same format as in the credentials Secret.

   The token is given in the `access_token` field of the response.

2. Get the list of vNIC profiles:

   ```
   curl -s -H "Authorization: Bearer <TOKEN>" <ZVIRT_ENGINE_URL>/api/vnicprofiles
   ```

   Replace `<TOKEN>` with the token from the previous step.

The profile ID is given in the `id` attribute of the `vnic_profile` element. Response example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vnic_profiles>
   <vnic_profile href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-8594eafd5a86"
id="49bb4594-0cd4-4eb7-8288-8594eafd5a86">
      <name>vm-net-01</name>
      <link href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-8594eafd5a86/permissions"
rel="permissions"/>
      <pass_through>
         <mode>disabled</mode>
      </pass_through>
      <port_mirroring>false</port_mirroring>
      <network href="/ovirt-engine/api/networks/74a741c9-0d40-4008-8e58-1c903ee6eba7" id="74a741c9-0d40-
4008-8e58-1c903ee6eba7"/>
   </vnic_profile>
   ...
</vnic_profiles>
```

How to disable individual provider subsystems?

To disable the nodes, storage, or Cloud Controller Manager subsystem, in the ModuleConfig resource of the `cloud-provider-zvirt` module, set the `nodes.disabled`, `storage.disabled`, or `ccm.disabled` parameter to `true`.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

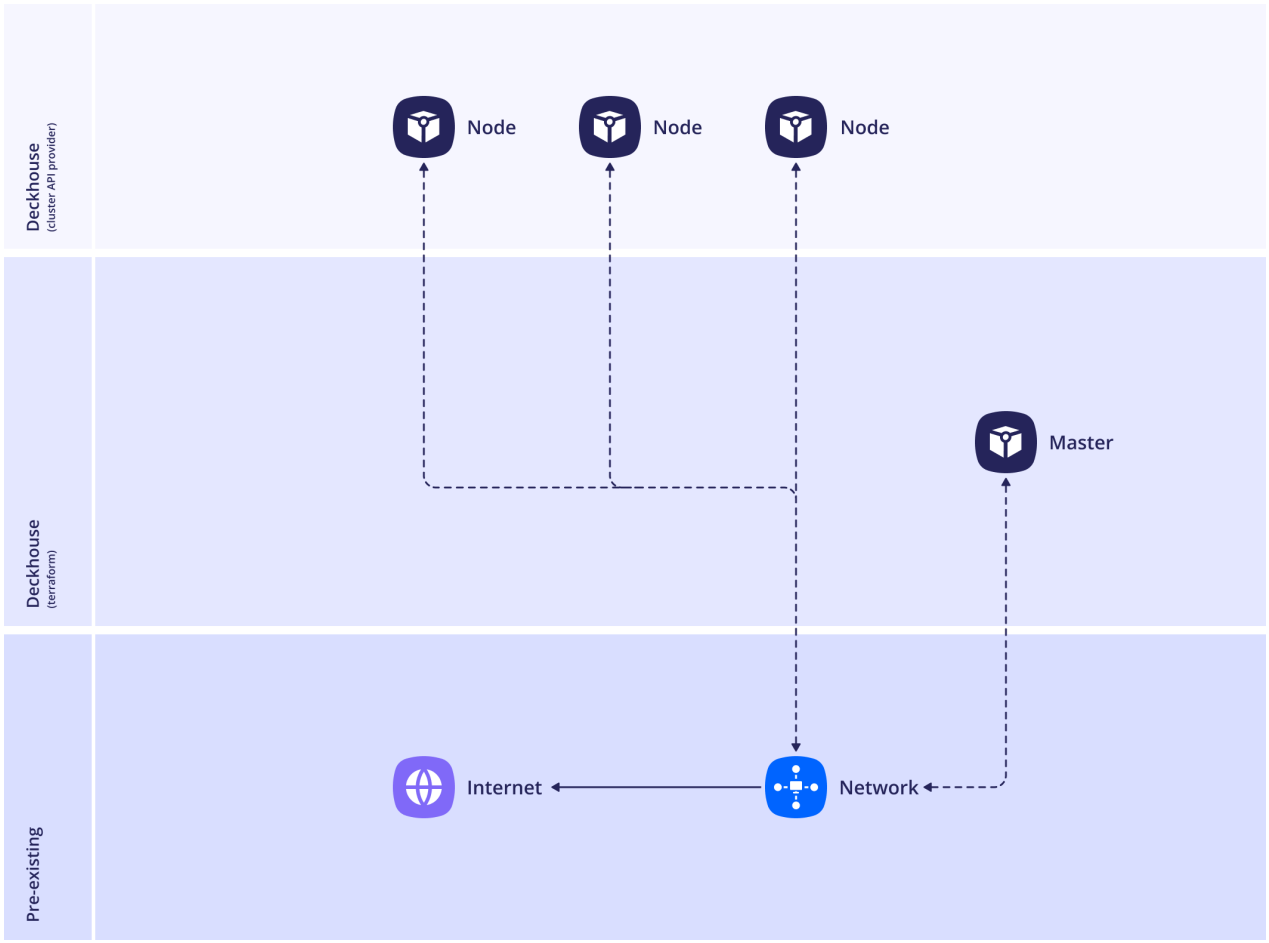
Layouts

> **i** The module lifecycle stage: General Availability

Only the Standard layout is supported. The layout is set with the `layout` parameter in the ModuleConfig resource of the module.

Standard

In the Standard layout, all cluster nodes are created in a single zVirt cluster set with the `clusterID` parameter. Virtual machines are cloned from a prepared template, and their network interface is created from a vNIC profile.



Example of the layout configuration with one master node:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 2
  enabled: true
  settings:
    provider:
      parameters:
        server: <ZVIRT_API_URL>
        clusterID: <CLUSTER_ID>
    nodes:
      parameters:
        layout: Standard
        sshPublicKey: <SSH_PUBLIC_KEY>
---
apiVersion: v1
kind: Secret
metadata:
  name: d8-credentials
  namespace: d8-cloud-provider-zvirt
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: userPassword
  identity: <USERNAME>
  secret: <PASSWORD>
---
apiVersion: deckhouse.io/v1
kind: ZvirtInstanceClass
metadata:
  name: master
spec:
  numCPUs: 4
  memory: 8192
  rootDiskSizeGb: 50
  etcdDiskSizeGb: 10
  template: <TEMPLATE_NAME>
  vnicProfileID: <VNIC_PROFILE_ID>
  storageDomainID: <STORAGE_DOMAIN_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: CloudPermanent
  cloudInstances:
    classReference:
      kind: ZvirtInstanceClass
      name: master
    minPerZone: 1
    maxPerZone: 1
```

Where:

- `ZVIRT_API_URL`: URL of the zVirt API endpoint, for example, `https://zvirt.example.com/ovirt-engine/api`.
- `CLUSTER_ID`: ID of the zVirt cluster to create virtual machines in.
- `SSH_PUBLIC_KEY`: Public SSH key for accessing the nodes.
- `USERNAME` and `PASSWORD`: zVirt user login and password for the Secret whose format is covered in the “Credentials Secret” section.
- `TEMPLATE_NAME`: Name of the virtual machine template prepared as described in the “Prepare a virtual machine template” section.
- `VNIC_PROFILE_ID`: ID of the vNIC profile that you can get as described in the “How to get the vNIC profile ID?” section.
- `STORAGE_DOMAIN_ID`: ID of the zVirt storage domain to create the etcd disk in.

The number of master nodes is set with the `minPerZone` parameter of the `master` node group. In the ZvirtInstanceClass resource of the `master` group, set the `etcdDiskSizeGb` and `storageDomainID` parameters, which set the size and the storage domain of a separate disk for etcd. The `etcdDiskSizeGb` parameter cannot be set in the instance classes of other CloudPermanent groups.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The cni-cilium module

> **i** Available with limitations in: **Open/CE, Core, BE, SE**
>
> Available without limitations in: **SE+, Ultimate/EE**

> **i** Included with limitations in extensions: **Advanced Networking**
>
> Included without limitations in extensions: **Cluster Union**

> **i** The module lifecycle stage: General Availability

The `cni-cilium` module provides a network in a cluster. It is based on the Cilium project.

Limitations

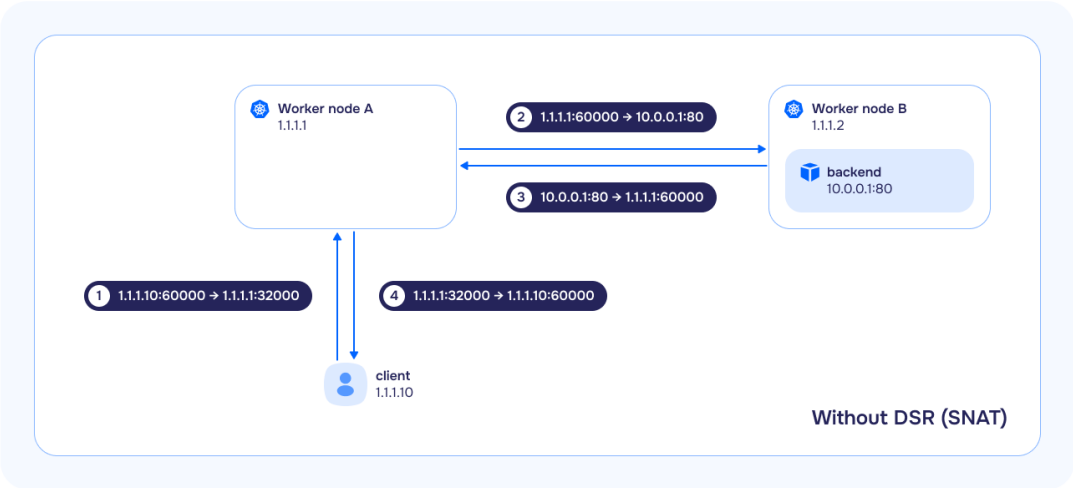
1. Services with type `NodePort` and `LoadBalancer` are incompatible with hostNetwork endpoints in LB mode `DSR`. Switch to `SNAT` mode if it is required.
2. `HostPort` pods only bind to one IP address. If the OS has multiple interfaces/IP, Cilium will choose one, preferring `private` to `public`.
3. If a node's `ExternalIP` is not assigned to any of the node's network interfaces but is provided by external infrastructure (for example, via 1:1 NAT), traffic from the PodNetwork to that `ExternalIP` is not supported. As a result, Pods that do not use `hostNetwork` cannot access a port exposed via `hostPort` using the node's `ExternalIP`. Use the node's `InternalIP` for such connections.
4. To ensure the stable operation of `cni-cilium` on cluster nodes, disable Elastic Agent or restrict its access to the Elastic management server. Elastic Agent includes an Elastic Endpoint component which uses the Extended Berkeley Packet Filter (eBPF) technology on cluster nodes and may remove critical eBPF programs required for `cni-cilium` to work correctly. For detailed information and discussion of the issue, refer to the publications of the Cilium and Elastic projects.
5. Kernel requirements:
   - Linux kernel version not lower than `5.8` for the `cni-cilium` module to work and work together with the istio, openvpn or node-local-dns modules.
6. OS compatibility:
   - Ubuntu:
     - incompatible with version 18.04;
     - HWE kernel installation required for working with version 20.04.
   - Astra Linux:
     - incompatible with the "Smolensk" edition.
   - CentOS:
     - for versions 7 and 8, a new kernel from the repository is required.

Handling external traffic in different `bpfLB` modes (replacing kube-proxy from Cilium)

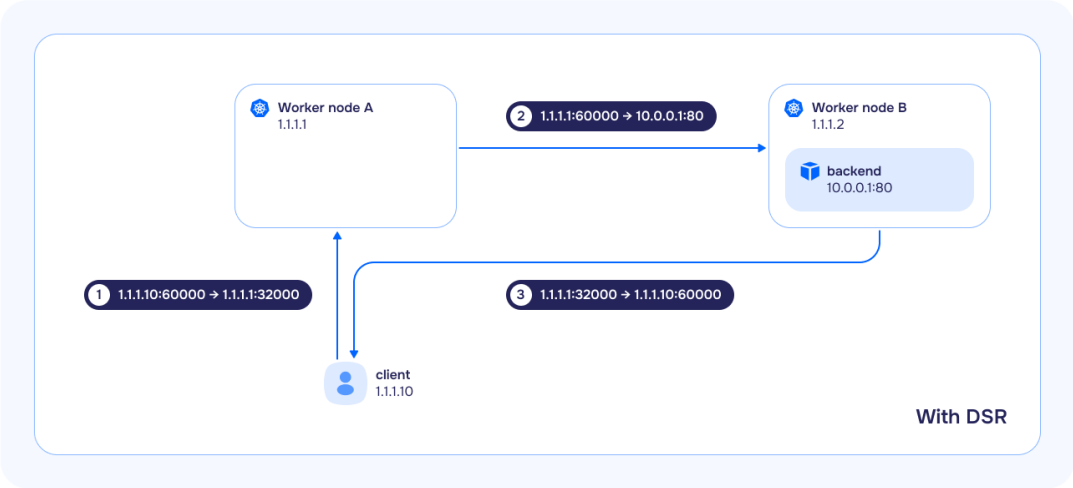
Kubernetes typically uses schemes where traffic comes to a balancer that distributes it among many servers. Both incoming and outgoing traffic passes through the balancer. Thus, the total throughput is limited by the resources and channel width of the balancer. To optimize traffic and unload the balancer, the `DSR` mechanism was invented, in which incoming packets go through the balancer, and outgoing ones go directly from the terminating servers. Since responses are usually much larger in size than requests, this approach can significantly increase the overall throughput of the scheme.

To extend the capabilities, the module allows selectable mode of operation, which affects the behavior of `Service` with the `NodePort` and `LoadBalancer` types:

- `SNAT` (Source Network Address Translation) — is a subtype of NAT in which, for each outgoing packet, the source IP address is translated to the IP address of the gateway from the target subnet, and incoming packets passing through the gateway are translated back based on a translation table. In this mode, `bpfLB` fully replicates the logic of `kube-proxy`:
  - if `externalTrafficPolicy: Local` is specified in the `Service`, the traffic will be forwarded and balanced only to those target pods running on the same node where the traffic arrived. If the target pod is not running on this node, the traffic will be dropped.
  - if `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. At the same time, if the target pods are located on other nodes, SNAT will be performed when transmitting traffic to them (the source IP address will be replaced with the InternalIP of the node).



- `DSR` (Direct Server Return) — is a method where all incoming traffic passes through the load balancer, and all outgoing traffic bypasses it. This method is used instead of `SNAT`. Often, responses are much larger than requests, and `DSR` can significantly increase the overall throughput of the scheme:
  - if `externalTrafficPolicy: Local` is specified in the `Service`, its behavior is completely analogous to `kube-proxy` and `bpfLB` in `SNAT` mode.
  - if `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. It is important to take into account the following features:
    - if the target pods are on other nodes, then the source IP address will be preserved when incoming traffic is sent to them;
    - outgoing traffic will go directly from the node on which the target pod was launched;
    - the source IP address will be replaced with the external IP address of the node to which the incoming request originally came.



⚠ In case of using `DSR` and `Service` mode with `externalTrafficPolicy: Cluster` additional network environment settings are required. Network equipment must be ready for asymmetric traffic flow: IP address anti-spoofing tools (`uRPF`, `sourceGuard`, etc.) must be disabled or configured accordingly.

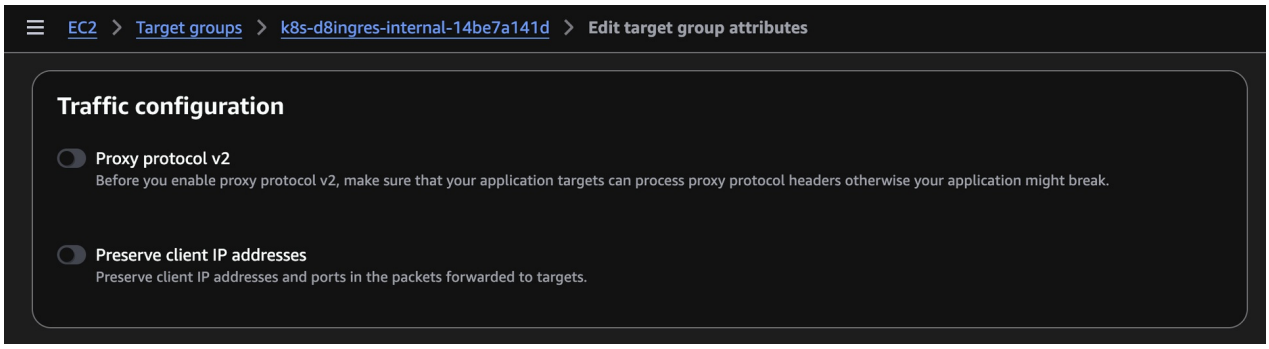
- `Hybrid` — in this mode, TCP traffic is processed in `DSR` mode, and UDP in `SNAT` mode.

Operational specifics of CNI Cilium on AWS

If Cilium runs in VXLAN mode (the only mode available on AWS), then by default connections from inside the cluster (from a node or a pod) to internal resources exposed through an `internal` load balancer (Load Balancer) do not work. From outside the cluster, the load balancer address answers. The cause is the enabled `preserve_client_ip` option (“Preserve client IP addresses”) on the target group: the load balancer skips NAT and the return traffic goes straight through the VXLAN tunnel past the load balancer. The traffic becomes asymmetric, and the connection drops. To fix it, disable `preserve_client_ip` on the target group in one of two ways.

⚠ After you disable `preserve_client_ip`, the load balancer performs SNAT, and the application stops seeing the real client IP address at the network level. For HTTP/HTTPS traffic this is not a problem: the address is still passed in the `X-Forwarded-For` header.

Using the AWS web console. Open **EC2 → Target groups**, select the load balancer target group (for example, `k8s-d8ingres-internal-...`), click **Edit target group attributes**, and in the **Traffic configuration** block turn off the **Preserve client IP addresses** toggle.



Using the AWS CLI. Run the command with your own target group ARN and region (you can get the ARN in the same **Target groups** section or with `aws elbv2 describe-target-groups`):

```
aws elbv2 modify-target-group-attributes \
  --target-group-arn arn:aws:elasticloadbalancing:REGION:ACCOUNT:targetgroup/NAME/ID \
  --attributes Key=preserve_client_ip.enabled,Value=false \
  --region REGION
```

Using CiliumClusterwideNetworkPolicy

✖ Applying CiliumClusterwideNetworkPolicy without `policyAuditMode` enabled may break control plane operation or cut SSH access to all cluster nodes.

The safe rollout procedure for CiliumClusterwideNetworkPolicy, including mandatory control plane and host firewall rules, is described in the Network policies section of the documentation. For details on the Cilium policy formats, refer to CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy.

Changing Cilium Operation Mode

When changing Cilium's operation mode (the tunnelMode parameter) from `Disabled` to `VXLAN` or vice versa, it is necessary to reboot all nodes, otherwise, pod availability issues may occur.

Disabling the kube-proxy Module

Cilium fully replaces the functionality of the `kube-proxy` module, so `kube-proxy` is automatically disabled when the `cni-cilium` module is enabled.

Using selective load balancing algorithm for services

In Deckhouse Platform, you can apply the following algorithms to load balance service traffic:

- `Random`: Randomly select a backend for each connection. Easy to implement, but does not always provide even distribution.
- `Maglev`: Uses consistent hashing to distribute traffic evenly, suitable for large-scale services.
- `Least Connections`: Directs traffic to the backend with the lowest number of active connections, optimizing load for applications with long-lived connections.

By default, the Random balancing algorithm is set for all services. However, Deckhouse allows you to override the algorithm for individual services. To use a selective balancing algorithm for a specific service, follow these steps:

- Edit the `cni-cilium` module configuration in Deckhouse by enabling the `extraLoadBalancerAlgorithmsEnabled` parameter. This activates support for service annotations for selective algorithms.
- In the service manifest, specify the `service.cilium.io/lb-algorithm` annotation with one of the values: `random`, `maglev`, or `least-conn`.

⚠ This mechanism requires Linux kernel version 5.15 or higher to work correctly.

Using Egress Gateway

Egress Gateway in Deckhouse Platform can be used in one of two modes: Basic mode and Virtual IP mode. Use Custom Resource EgressGateway (parameter `spec.sourceIP.mode`) to select the mode.

Basic mode

Pre-configured IP addresses are used on egress nodes.

Virtual IP mode

The ability to dynamically assign additional IP addresses to nodes is implemented.

Exporting data from Hubble

Deckhouse Platform allows to configure data export from Hubble running inside Cilium agents using the cluster-scoped custom resource HubbleMonitoringConfig. To enable export, create a HubbleMonitoringConfig resource.

⚠ Creating or modifying the HubbleMonitoringConfig resource will restart all Cilium agents in the cluster.

After the manifest is applied:

- Hubble metrics become available in Prometheus. All exported metrics have the `hubble_*` prefix;
- Hubble logs are written to the `/var/log/cilium/hubble/flow.log` file on each node.

Third-party components

List of third-party software used in the `cni-cilium` module:

- Cilium 1.17.17

  License: Apache License 2.0

  Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application workloads such as application containers or processes.

- CNI plugins 1.9.1

  License: Apache License 2.0

  Some reference and example networking plugins, maintained by the CNI team.

- Gops 0.3.27

  License: BSD 3-Clause License

  A tool to list and diagnose Go processes currently running on your system.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

Available with limitations in: **Open/CE, Core, BE, SE**

Available without limitations in: **SE+, Ultimate/EE**

Included with limitations in extensions: **Advanced Networking**

Included without limitations in extensions: **Cluster Union**

The module lifecycle stage: General Availability

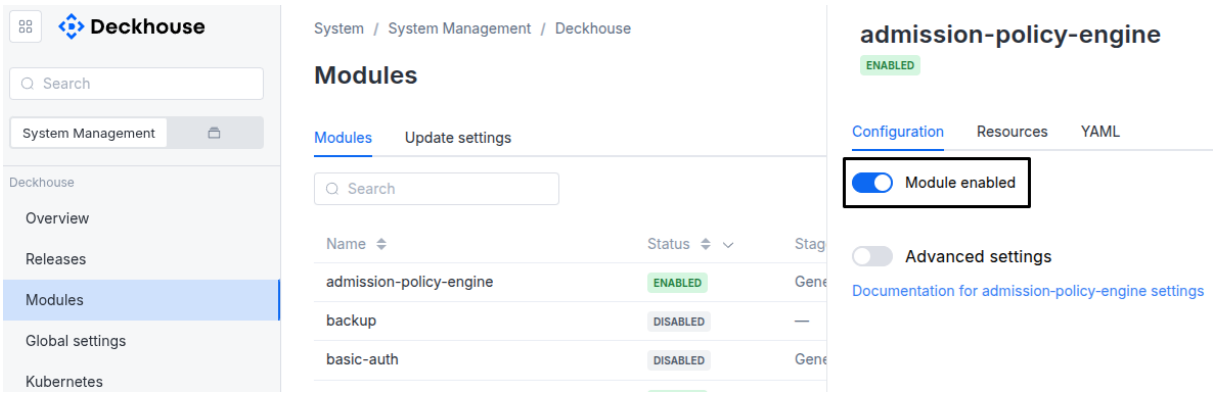
The module has 17 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-cilium` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module

(you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable cni-cilium
```

- **Using ModuleConfig `cni-cilium`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cni-cilium` (create it if necessary);

  Example of a manifest to enable module `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  enabled: true
```

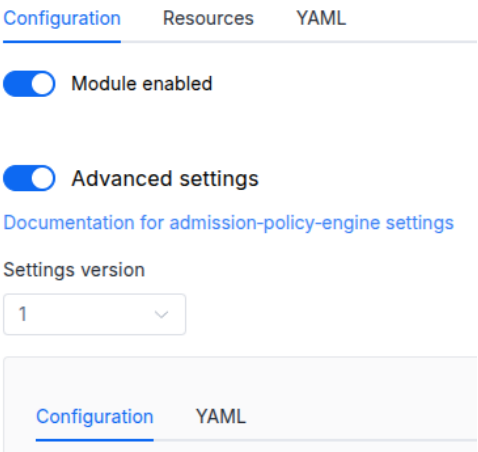
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-cilium` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cni-cilium` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cni-cilium`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cni-cilium` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cni-cilium
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cni-cilium` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.activeL2NeighborDiscoveryEnabled**
    boolean

    Enables the Cilium `L2 neighbor discovery` mechanism, which helps ensure that ARP table entries are kept up-to-date on all cluster nodes.

    Default: `true`

  - **settings.bpfLBMode**
    string

    eBPF LoadBalancer mode:

    - `SNAT` — traffic from the client to the pod (and back) passes through NAT, and accordingly the sender's address is lost.
    - `DSR` — traffic from the client to the pod passes with the sender's address preserved, and back - according to the routing rules (bypassing the balancer). This mode saves network traffic and reduces delays, but only works for TCP traffic.
    - `Hybrid` — TCP traffic is processed in DSR mode, and UDP traffic is processed in SNAT mode.

    Default: `Hybrid`

    Allowed values: `SNAT`, `Hybrid`, `DSR`

  - **settings.createNodeRoutes**
    boolean

    Create routes to Pods on other Nodes.

    All Nodes must be located in the same L2 domain.

  - **settings.debugLogging**
    boolean

    Enables debug logging for Cilium components.

    Default: `false`

  - **settings.encryption**
    object

    CNI Cilium encryption settings.

    - **settings.encryption.mode**
      string

      Encryption mode:

      - `Disabled` — encryption is disabled.
      - `WireGuard` — enables traffic encryption between pods using WireGuard.

        > **i** Note! Before using WireGuard encryption, please ensure that the Linux kernel running on the cluster nodes has support WireGuard (either via the CONFIG_WIREGUARD=m kernel config option on Linux 5.6 and newer, or using an external WireGuard module for older kernels).

        > **i** Note! Using WireGuard may cause significant CPU consumption by Cilium agent pods.

      Default: `Disabled`

      Allowed values: `Disabled`, `WireGuard`

    - **settings.encryption.nodeToNodeModeEnabled**
      boolean

      Enabling traffic encryption between nodes and pods in HostNetwork.

      > **i** Note! The node-to-node traffic from and to the control-plane nodes is left unencrypted to prevent possible communication issues when renewing WireGuard public keys. At the same time, the traffic between control-plane components is still encrypted at the application layer.

      > **i** Note! When this mode is active, `NodePort` and `HostPort` are not available from the `PodNetwork`.

      Default: `false`

  - **settings.exclusiveCNIPlugin**
    boolean

    Prevents additional CNI plugins from being enabled on cluster nodes. If the Istio module's `dataPlane.trafficRedirectionSetupMode` setting is set to `CNIPlugin`, the option will be automatically set to

`false`.

Explicitly disabling this option is recommended only when using an additional CNI or a specific CNI plugin in the cluster. An example of such a use case could be the Linkerd CNI plugin.

Default: `true`

- **settings.extraLoadBalancerAlgorithmsEnabled**
  boolean

  Allows selectively overriding the load balancing algorithm using eBPF between backends for individual services. The annotation `service.cilium.io/lb-algorithm` is used for overriding with one of the following values: `random`, `maglev`, or `least-conn`.

  > **i** Note! This option requires the Linux kernel version on the nodes to be at least 5.15.

  Default: `false`

- **settings.ipv4OptionsGuard**
  object

  Mitigation of CVE-2026-98432 (use-after-free in the Linux IPv4 TCP handshake triggered by a TCP SYN with IP options).

  A sidecar of the CNI agent installs a filter into the host network namespace and into every pod network namespace on the node, including pods started later. The filter matches IPv4 TCP SYN packets and IPv4 fragments carrying IP options.

  The filter follows the iptables mode (nft or legacy) that iptables-wrapper selects on the node, the same as the other Deckhouse components. In the nft mode, the filter uses native nftables expressions. In the legacy mode, it uses iptables with the `u32` match. No kernel modules are loaded unless `allowKernelModuleLoad` is set. If the kernel modules of the selected mode are not loaded but those of the other mode are, the filter uses the other mode, and the node is reported by the `D8IPv4OptionsGuardBackendMismatch` alert. A node that cannot be protected is reported by the `D8IPv4OptionsGuardNodeUnprotected` alert.

  To exclude a node (for example, a node with CIPSO labeled networking), set the `ipv4-options-guard.deckhouse.io/disabled: "true"` label on it: the changes are reverted on that node.

  The mitigation does not fix the vulnerability: update the node kernel as soon as a fixed version is available.

  - **settings.ipv4OptionsGuard.allowKernelModuleLoad**
    boolean

    The filter needs the `nf_tables` kernel module in the nft mode and the `ip_tables`, `iptable_raw`, `xt_tcpudp` and `xt_u32` kernel modules in the legacy mode.

    By default, kernel modules are not loaded: if a module of the mode selected by iptables-wrapper is not loaded, the filter uses the other mode when its modules are loaded, and the node is reported as unprotected otherwise. Set to `true` to load the modules of the selected mode.

    Default: `false`

  - **settings.ipv4OptionsGuard.disableMPTCP**
    boolean

    In the `Enforce` mode, set `net.mptcp.enabled=0` in the host and in every pod network namespace. This removes the precondition of the published exploit.

    Listeners opened before the change stay MPTCP until the process is restarted. After a node reboot, MPTCP on the host is disabled again once the CNI agent starts.

    Default: `true`

  - **settings.ipv4OptionsGuard.mode**
    string

    Operating mode.

    If the mode is not set, it is `Enforce` in clusters installed with the mitigation and `Audit` in clusters that existed before it.

    - `Audit` — matching packets are only counted (the `d8_ipv4_options_guard_matched_packets_total` metric and the `D8IPv4OptionsGuardMatchedTraffic` alert). Use it to check that no workload relies on IP options. While any node is in this mode, the `D8IPv4OptionsGuardAuditMode` alert reminds that the cluster is not protected yet.
    - `Enforce` — matching packets are dropped, and MPTCP is disabled if `disableMPTCP` is set.
    - `Disabled` — the filter is removed from all network namespaces, including running pods, and `net.mptcp.enabled` is restored to the value it had before.

    The mode is applied without restarting the CNI agent pods.

    In the `Audit` and `Enforce` modes with `bpfLBMode` set to `DSR` or `Hybrid`, the backend node also removes the DSR IPv4 option from service packets before they reach the pod. Switching to or from `Disabled` turns this on or off and restarts the Cilium agents.

    Allowed values: `Audit`, `Enforce`, `Disabled`

- **settings.labelsRegex**
  array of strings

  Cilium creates security identifiers based on the k8s entity labels. The more labels are involved in this process, the higher the access granularity that can be achieved. However, in large clusters, excessive granularity can create a

heavy load. You can use the `labelsRegex` option to explicitly specify which labels are to be used for security policies, and which labels are to be neglected. Refer to the documentation to learn more about reducing the identity cardinality

Each label must be specified by a RegExp expression in YAML quoted string format. Note that special characters must be escaped.

Example:

```
labelsRegex:
- k8s:!app\.kubernetes\.io
- k8s:io\.cilium\.k8s\.policy
```

- **settings.loadBalancerICMPReplyEnabled**
  boolean

  Enable ICMP Echo Reply generation for LoadBalancer service IPs.

  When enabled, nodes respond to ICMP Echo Requests (ping) addressed to LoadBalancer external IPs.

  Default: `false`

- **settings.loadBalancerSourceRangeAllTypes**
  boolean

  Enable loadBalancerSourceRanges CIDR filtering for all service types, not just LoadBalancer services. The corresponding NodePort and ClusterIP will also apply the CIDR filter. For more detailed information, please visit the link.

  Default: `true`

- **settings.masqueradeMode**
  string

  Cilium masquerade work mode for pods traffic leaving the cluster.

  - `BPF` - use cilium BPF. Basic operation mode.

    In this mode, masquerade will not be used if the destination IP address is within the `podSubnetCIDR` or `InternalIP/ExternalIP` of any of the cluster nodes.

  - `Netfilter` - use kernel Netfilter(iptables/nf_tables).

    In this mode, masquerade will not be used if the destination IP address is within the `podSubnetCIDR` or `InternalIP/ExternalIP` of any of the cluster nodes.

  > **i** In `BPF` mode, if `ExternalIP` other than `InternalIP` is specified for a cluster node, and that IP address is not currently assigned to any interface on the node, it will not be reachable from neighboring nodes from pods network. The `Netfilter` mode allows you to bypass this limitation, but some other features stop working, such as `EgressGateway`.

  Default: `BPF`

  Allowed values: `Netfilter`, `BPF`

- **settings.networkPeering**
  object

  Available in editions: Ultimate/EE

  Experimental.

  NetworkPeering: direct pod-to-pod connectivity with the clusters this cluster is peered with by the `peering` module, built on Cilium ClusterMesh.

  A peer is connected when its `ClusterPeer` lists the `network` capability on both sides; the module shows the state of the mesh with it in a `ClusterNetworkPeer` object. Requires `tunnelMode: VXLAN` and `encryption.mode: Disabled` on every cluster of the mesh, the same VXLAN port, and non-overlapping pod subnets.

  The cluster name in the mesh is the cluster UUID without dashes (32 characters).

  > **i** The cluster name and ID become part of every Cilium identity. Enabling NetworkPeering on a running cluster reallocates identities: plan pod restarts.

  - **settings.networkPeering.clusterID**
    integer

    Cluster ID in the mesh. Must be unique among all clusters this cluster is meshed with, and among their own peers.

    Allowed values: `1 <= X <= 255`

    Example:

    ```
    clusterID: 11
    ```

  - **settings.networkPeering.enabled**
    boolean

Enable NetworkPeering.

Default: `false`

- **settings.networkPeering.hostPort**
  object

  Settings for the `HostPort` inlet.

  - **settings.networkPeering.hostPort.port**
    integer

    Host port of the etcd of `clustermesh-apiserver` on control-plane nodes. Must be reachable from the control-plane nodes of the peer clusters, where their kvstoremesh runs. Must not collide with another host port of the platform. Must not be 2379, which the control-plane etcd uses.

    Default: `4294`

    Allowed values: `1024 <= X <= 65535`

    Example:

    ```
    port: 4294
    ```

- **settings.networkPeering.inlet**
  string

  How `clustermesh-apiserver` is exposed to the peers.

  - `HostPort`: one replica per control-plane node, each on a host port. A peer connects to one replica at a time and moves to another only when that one fails, so a reconnection does not land on a different etcd instance at random.

  Default: `HostPort`

  Allowed values: `HostPort`

- **settings.nodePortAddresses**
  array of strings

  List of CIDRs that restrict which node IP addresses are used for Cilium BPF NodePort listeners.

  Only addresses assigned to selected network devices and matching one of the configured CIDRs will be exposed as NodePort frontends.

  Example:

  ```
  nodePortAddresses:
  - 192.168.1.100/32
  - 10.0.0.0/8
  - 192.168.1.0/24
  ```

- **settings.policyAuditMode**
  boolean

  Do not enforce any Network Policy. Just log allowed/disallowed connections.

  Default: `false`

- **settings.resourcesManagement**

  Settings for CPU and memory requests and limits by cilium agent pods.

  Examples:

  ```
  resourcesManagement:
    mode: VPA
    vpa:
      mode: InPlaceOrRecreate
      cpu:
        min: 50m
        max: 2
        limitRatio: 1.5
      memory:
        min: 256Mi
        max: 2Gi
        limitRatio: 1.5
  ```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

- **settings.resourcesManagement.mode**
  string

  Resource management mode:

  - `Static` is a classic one. In it, you specify requests/limits. The parameters of this mode are defined in the static parameter section;
  - `VPA` mode uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

  Default: `VPA`

  Allowed values: `VPA`, `Static`

- **settings.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.resourcesManagement.static.limits.cpu**
      string or number

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.resourcesManagement.static.requests.cpu**
      string or number

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.resourcesManagement.vpa.cpu.limitRatio**

number

The CPU limits/requests ratio.

This ratio is used for calculating the initial CPU limits for a pod.

If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

- **settings.resourcesManagement.vpa.cpu.max**
  string or number

  The maximum value that the VPA can set for the CPU requests.

  Default: `4`

  Pattern: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.cpu.min**
  string or number

  The minimum value that the VPA can set for the CPU requests.

  Default: `100m`

  Pattern: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.memory**
  object

  Memory-related VPA settings.

  - **settings.resourcesManagement.vpa.memory.limitRatio**
    number

    The memory limits/requests ratio.

    This ratio is used for calculating the initial memory limits for a pod.

    If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

  - **settings.resourcesManagement.vpa.memory.max**
    string or number

    The maximum memory requests the VPA can set.

    Default: `4Gi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesManagement.vpa.memory.min**
    string or number

    The minimum memory requests the VPA can set.

    Default: `128Mi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.mode**
  string

  VPA operating mode.

  - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
  - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
  - `Auto` — VPA automatically recreates pods to apply updated resource values.

    > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

  Default: `Initial`

  Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.sctpEnabled**
  boolean

  Enable SCTP support for Cilium (for both incoming and outgoing traffic).

  > **i** **Note.** Cilium implements only basic SCTP support. Therefore, when using SCTP with Cilium, please be aware of the following limitations:

  - The following features are not currently supported: `SCTP multi-homing`, `NetworkPolicies for pod-to-VIP`.
  - For incoming SCTP packets port overwriting is not supported. This means that applications must be published through either HostPort or Service, where the `targetPort` MUST match the `port` specified. Otherwise, the packet will be dropped.

    - For outgoing SCTP packets, only 1:1 SNAT is supported by default. In this mode, only the src_ip is replaced, while the src_port remains unchanged. If the src_port is already in use on the node, the packet will be rejected. To enable src_port substitution, you must switch the `masqueradeMode` parameter to `Netfilter` mode.

  Default: `false`

- **settings.svcSourceRangeCheck**
  boolean

  For services of the `loadBalancer` type, enables checking the source IP for compliance with loadBalancer.sourceRanges. The check is not performed if access is made from within the cluster or the `sourceRanges` parameter is empty/not specified. Some cloud providers support filtering of incoming traffic based on the `sourceRanges` parameter in their loadBalancer implementations. In such cases, the cilium documentation recommends disabling the `svcSourceRangeCheck` option.

  Do not enable if another balancer or any type of SNAT is running in front of the services.

  Default: `false`

- **settings.tunnelMode**
  string

  Tunnel mode.

  > **i** Note! After changing the parameter, it is necessary to restart all nodes, otherwise, there may be problems with the availability of Pods!

  > **i** Note! VXLAN mode is not compatible with following bpfLBMode modes: Hybrid, DSR. bpfLBMode will be overriden to SNAT if the tunnelMode is VXLAN.

  Default: `Disabled`

  Allowed values: `Disabled`, `VXLAN`

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> **i** The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

EgressGatewayPolicy

Scope: Cluster
Version: v1alpha1

Interface for configuring policies to redirect application traffic to specific egress gateways described through the EgressGateway interface.

- **spec**
  object

  Required value

  - **spec.destinationCIDRs**
    array of strings

    Available in editions: SE+, Ultimate/EE

    Network requests towards these subnets will be routed through the egress gateway.

  - **spec.egressGatewayName**
    string

    Required value

    Available in editions: SE+, Ultimate/EE

    The name of the EgressGateway resource that describes the egress gateway for handling traffic.

  - **spec.excludedCIDRs**
    array of strings

    Available in editions: SE+, Ultimate/EE

    Network ranges that are required to be excluded from the ranges described in `destinationCIDRs`. Note that you do not need to exclude the special cluster ranges (`podSubnetCIDR` and `serviceSubnetCIDR`).

- **spec.selectors**
  array of objects

  Available in editions: SE+, Ultimate/EE

  Application Pod Selectors. Their traffic will be routed through the appropriate egress gateway. For namespase designation use the `io.kubernetes.pod.namespace` pseudo-label.

  - **spec.selectors.podSelector**
    object

    Required value

    Available in editions: SE+, Ultimate/EE

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **spec.selectors.podSelector.matchExpressions**
      array of objects

      Available in editions: SE+, Ultimate/EE

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **spec.selectors.podSelector.matchExpressions.key**
        string

        Required value

        Available in editions: SE+, Ultimate/EE

        key is the label key that the selector applies to.

      - **spec.selectors.podSelector.matchExpressions.operator**
        string

        Required value

        Available in editions: SE+, Ultimate/EE

        operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **spec.selectors.podSelector.matchExpressions.values**
        array of strings

        Available in editions: SE+, Ultimate/EE

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **spec.selectors.podSelector.matchLabels**
      object

      Available in editions: SE+, Ultimate/EE

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **status**
  object

EgressGateway

Scope: Cluster
Version: v1alpha1

Interface for configuring a fault-tolerant egress gateway based on a group of nodes.

The configured egress gateways can be used in EgressGatewayPolicy interface to configure egress gateway for application Pods.

- **spec**
  object
  - **spec.nodeSelector**
    object

    Available in editions: SE+, Ultimate/EE

    The selector for a group of nodes that will transfer network requests to external services.

    Among these nodes, eligible nodes will be detected and one of them will be assigned as the active one. Signs of an eligible node:

    - The node is in Ready state.
    - Node is not cordoned.
    - The cilium-agent on the node is in the Ready state.

    Different EgressGateways can use common nodes for operation, and active nodes will be selected independently for

each EgressGateway, thus distributing the load between them.

- **spec.sourceIP**
  object
  - **spec.sourceIP.mode**
    string

    Required value

    Available in editions: SE+, Ultimate/EE

    A method for determining the source IP address to be assigned to requests through this gateway.

    Possible options:

    - `PrimaryIPFromEgressGatewayNodeInterface` (basic mode) — the primary IP address on the *public* network interface of the node will be used as the source IP address.

      Particularities:

      - When the active node goes down and a new node is assigned, the outgoing IP address will change.
      - The network subsystem on all egress nodes must be configured in advance:
        - the *public* interfaces must have the same name (eth1, for example),
        - all necessary routes for access to all external public services must be configured.

    - `VirtualIPAddress` (Virtual IP mode) — explicitly specify the outgoing IP address.

      Particularities:

      - If the active node goes down, the outgoing IP address will not change.
      - The network subsystem on all egress nodes must be configured in advance:
        - all necessary routes for access to all external public services must be configured,
        - the *public* interface must be prepared to automatically configure a *virtual* IP as a secondary IP address (if an egress node is designated as active, the address will not appear in the IP list on the *public* interface, but the node will emulate it with ARP-responses)

    Allowed values: `VirtualIPAddress`, `PrimaryIPFromEgressGatewayNodeInterface`

  - **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface**
    object

    Used for basic mode (if `mode: PrimaryIPFromEgressGatewayNodeInterface`).

    - **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface.interfaceName**
      string

      Available in editions: SE+, Ultimate/EE

      The interface name on the egress nodes through which external requests are processed.

  - **spec.sourceIP.virtualIPAddress**
    object

    Used for mode with Virtual IP (if `mode: PrimaryIPFromEgressGatewayNodeInterface`).

    - **spec.sourceIP.virtualIPAddress.interfaces**
      array of strings

      Available in editions: SE+, Ultimate/EE

      The list of network interfaces to which the *virtual* IP will be simulated.

      Default: `[]`

    - **spec.sourceIP.virtualIPAddress.ip**
      string

      Available in editions: SE+, Ultimate/EE

      The *virtual* source IP address to be assigned to outgoing requests through the egress gateway.

- **status**
  object
  - **status.activeNodeName**
    string

    The name of the current active node.

  - **status.conditions**
    array of objects
    - **status.conditions.lastHeartbeatTime**
      string
    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

message is a human readable message indicating details about the transition. This may be an empty string.

Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Type of condition in CamelCase or in foo.example.com/CamelCase.

  Many .condition.type values are consistent across resources like Available, but because arbitrary conditions can be useful (see .node.status.conditions), the ability to deconflict is important. The regex it matches is (dns1123SubdomainFmt/)?(qualifiedNameFmt)

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.observedGeneration**
  integer
- **status.readyNodes**
  integer

  Number of nodes ready to perform egress gateway functions.

HubbleMonitoringConfig

Scope: Cluster
Version: v1alpha1

- **spec**
  object

  Required value

  - **spec.extendedMetrics**
    object

    Configuration for exporting extended metrics from Cilium agents.

    Extended metrics provide deeper insight into cluster network behavior.

    > **Warning.** Changing this parameter will trigger a restart of the Cilium agents.

    For more details, see the Cilium documentation.

    Example:

```
extendedMetrics:
  enabled: true
  collectors:
  - name: drop
    contextOptions:
labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,destination_pod
  - name: flow
```

- **spec.extendedMetrics.collectors**
  array of objects

  List of Hubble extended metrics to export.

  Each collector represents a specific metric type.

  For more details, see the Cilium documentation.

  Example:

```
collectors:
- name: drop
  contextOptions:
labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,de
stination_pod
- name: flow
```

  - **spec.extendedMetrics.collectors.contextOptions**
    string

    Optional context options for Hubble metrics.

    Defines which parameters are used to break down the metrics produced by a collector. If you do not specify `contextOptions`, the collector returns the default set of metrics aggregated overall, without breakdown by individual parameters. For example, for the `drop` collector, an empty `contextOptions` returns overall drop statistics without distinguishing the exact source and destination. When you set `contextOptions`, metric values are grouped by the selected dimensions, for example by source and destination Pods, namespaces, and labels.

    > **Warning.** A large number of labels on metrics can lead to Prometheus performance degradation.

    For more information about supported options, see the Cilium documentation.

  - **spec.extendedMetrics.collectors.name**
    string

    Name of the extended metric to export.

    Allowed values: `dns`, `drop`, `flow`, `flows-to-world`, `http`, `httpV2`, `icmp`, `kafka`, `tcp`

- **spec.extendedMetrics.enabled**
  boolean

  Enables the export of extended metrics from Cilium agents.

  > All exported metrics have the prefix `hubble_*`.

  Default: `false`

- **spec.flowLogs**
  object

  Configuration for exporting event logs from Cilium agents.

  Event logs provide detailed visibility into Cilium's network events.

  > **Warning.** Changing this parameter will trigger a restart of the Cilium agents.

  For more details see the Cilium documentation.

  Example:

```
flowLogs:
  enabled: true
  allowFilterList:
  - verdict:
    - DROPPED
    - ERROR
  denyFilterList:
  - source_pod:
    - kube-system/
  - destination_pod:
    - kube-system/
  fieldMaskList:
  - time
  - source.namespace
  - verdict
  fileMaxSizeMB: 50
```

  - **spec.flowLogs.allowFilterList**

array of objects

Defines filters that control which events are exported for logging. Only events that match at least one of the specified filter objects will be exported. If no filters are specified, all events are allowed. Logical semantics: - Within a single filter object, all specified fields are combined using AND. - Across filter objects in the array, filters are combined using OR. For more details, see the Cilium documentation.

- **spec.flowLogs.allowFilterList.destination_cluster_name**
  array of strings

  Destination cluster names.

- **spec.flowLogs.allowFilterList.destination_fqdn**
  array of strings

  Destination fully qualified domain names (FQDN).

  - **Element of the array**
    string

    Maximum length: `255`

- **spec.flowLogs.allowFilterList.destination_identity**
  array of integers

  Destination security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.destination_ip**
  array of strings

  Destination IP or CIDR.

- **spec.flowLogs.allowFilterList.destination_label**
  array of strings

  Destination label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  destination_label: k8s:role=frontend
  ```

  ```
  destination_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.allowFilterList.destination_pod**
  array of strings

  Destination Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  destination_pod: kube-system/
  ```

  ```
  destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.allowFilterList.destination_port**
  array of strings

  L4 destination port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.destination_service**
  array of strings

  Destination service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

  ```
  destination_service: kube-system/kube-dns
  ```

  ```
  destination_service: kube-system/
  ```

  ```
  destination_service: "/kube-dns"
  ```

- **spec.flowLogs.allowFilterList.destination_workload**
  array of objects

Destination workloads.

- **spec.flowLogs.allowFilterList.destination_workload.kind**
  string

  Workload kind.

- **spec.flowLogs.allowFilterList.destination_workload.name**
  string

  Workload name.

- **spec.flowLogs.allowFilterList.dns_query**
  array of strings

  DNS query regex patterns.

- **spec.flowLogs.allowFilterList.drop_reason_desc**
  array of strings

  Drop reason (only for DROPPED verdict).

  - **Element of the array**
    string

    Allowed values: `DROP_REASON_UNKNOWN`, `INVALID_SOURCE_MAC`, `INVALID_DESTINATION_MAC`, `INVALID_SOURCE_IP`, `POLICY_DENIED`, `INVALID_PACKET_DROPPED`, `CT_TRUNCATED_OR_INVALID_HEADER`, `CT_MISSING_TCP_ACK_FLAG`, `CT_UNKNOWN_L4_PROTOCOL`, `CT_CANNOT_CREATE_ENTRY_FROM_PACKET`, `UNSUPPORTED_L3_PROTOCOL`, `MISSED_TAIL_CALL`, `ERROR_WRITING_TO_PACKET`, `UNKNOWN_L4_PROTOCOL`, `UNKNOWN_ICMPV4_CODE`, `UNKNOWN_ICMPV4_TYPE`, `UNKNOWN_ICMPV6_CODE`, `UNKNOWN_ICMPV6_TYPE`, `ERROR_RETRIEVING_TUNNEL_KEY`, `ERROR_RETRIEVING_TUNNEL_OPTIONS`, `INVALID_GENEVE_OPTION`, `UNKNOWN_L3_TARGET_ADDRESS`, `STALE_OR_UNROUTABLE_IP`, `NO_MATCHING_LOCAL_CONTAINER_FOUND`, `ERROR_WHILE_CORRECTING_L3_CHECKSUM`, `ERROR_WHILE_CORRECTING_L4_CHECKSUM`, `CT_MAP_INSERTION_FAILED`, `INVALID_IPV6_EXTENSION_HEADER`, `IP_FRAGMENTATION_NOT_SUPPORTED`, `SERVICE_BACKEND_NOT_FOUND`, `NO_TUNNEL_OR_ENCAPSULATION_ENDPOINT`, `FAILED_TO_INSERT_INTO_PROXYMAP`, `REACHED_EDT_RATE_LIMITING_DROP_HORIZON`, `UNKNOWN_CONNECTION_TRACKING_STATE`, `LOCAL_HOST_IS_UNREACHABLE`, `NO_CONFIGURATION_AVAILABLE_TO_PERFORM_POLICY_DECISION`, `UNSUPPORTED_L2_PROTOCOL`, `NO_MAPPING_FOR_NAT_MASQUERADE`, `UNSUPPORTED_PROTOCOL_FOR_NAT_MASQUERADE`, `FIB_LOOKUP_FAILED`, `ENCAPSULATION_TRAFFIC_IS_PROHIBITED`, `INVALID_IDENTITY`, `UNKNOWN_SENDER`, `NAT_NOT_NEEDED`, `IS_A_CLUSTERIP`, `FIRST_LOGICAL_DATAGRAM_FRAGMENT_NOT_FOUND`, `FORBIDDEN_ICMPV6_MESSAGE`, `DENIED_BY_LB_SRC_RANGE_CHECK`, `SOCKET_LOOKUP_FAILED`, `SOCKET_ASSIGN_FAILED`, `PROXY_REDIRECTION_NOT_SUPPORTED_FOR_PROTOCOL`, `POLICY_DENY`, `VLAN_FILTERED`, `INVALID_VNI`, `INVALID_TC_BUFFER`, `NO_SID`, `MISSING_SRV6_STATE`, `NAT46`, `NAT64`, `AUTH_REQUIRED`, `CT_NO_MAP_FOUND`, `SNAT_NO_MAP_FOUND`, `INVALID_CLUSTER_ID`, `UNSUPPORTED_PROTOCOL_FOR_DSR_ENCAP`, `NO_EGRESS_GATEWAY`, `UNENCRYPTED_TRAFFIC`, `TTL_EXCEEDED`, `NO_NODE_ID`, `DROP_RATE_LIMITED`, `IGMP_HANDLED`, `IGMP_SUBSCRIBED`, `MULTICAST_HANDLED`, `DROP_HOST_NOT_READY`, `DROP_EP_NOT_READY`, `DROP_NO_EGRESS_IP`

- **spec.flowLogs.allowFilterList.event_type**
  array of objects

  Event types to filter on.

  - **spec.flowLogs.allowFilterList.event_type.match_sub_type**
    boolean

    Whether to match on `sub_type`.

  - **spec.flowLogs.allowFilterList.event_type.sub_type**
    integer

    Secondary event type.

  - **spec.flowLogs.allowFilterList.event_type.type**
    integer

    Primary event type.

- **spec.flowLogs.allowFilterList.http_header**
  array of objects

  HTTP headers to match.

  - **spec.flowLogs.allowFilterList.http_header.key**
    string

    Header key.

  - **spec.flowLogs.allowFilterList.http_header.value**
    string

    Header value.

- **spec.flowLogs.allowFilterList.http_method**
  array of strings

  HTTP methods.

  - **Element of the array**
    string

    Allowed values: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`

- **spec.flowLogs.allowFilterList.http_path**
  array of strings

  Regex list to match HTTP path.

- **spec.flowLogs.allowFilterList.http_status_code**
  array of strings

  HTTP status code prefixes.

  - **Element of the array**
    string

    Pattern: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`

- **spec.flowLogs.allowFilterList.http_url**
  array of strings

  Regex list to match HTTP URL.

- **spec.flowLogs.allowFilterList.interface**
  array of objects

  Network interface where flow was observed.

  - **spec.flowLogs.allowFilterList.interface.index**
    integer

    Interface index.

    Allowed values: `0 <= X`

  - **spec.flowLogs.allowFilterList.interface.name**
    string

    Interface name.

- **spec.flowLogs.allowFilterList.ip_version**
  array of strings

  IP version.

  - **Element of the array**
    string

    Allowed values: `IP_NOT_USED`, `IPv4`, `IPv6`

- **spec.flowLogs.allowFilterList.node_labels**
  array of strings

  Node label selectors.

  Examples:

  ```
  node_labels: node-role.kubernetes.io/control-plane=
  ```

  ```
  node_labels: topology.kubernetes.io/ru-central1-a
  ```

- **spec.flowLogs.allowFilterList.node_name**
  array of strings

  Node name patterns.

- **spec.flowLogs.allowFilterList.protocol**
  array of strings

  L4 or L7 protocol.

- **spec.flowLogs.allowFilterList.source_cluster_name**
  array of strings

  Source cluster names.

- **spec.flowLogs.allowFilterList.source_fqdn**
  array of strings

  Source fully qualified domain names (FQDN).

  - **Element of the array**
    string

    Maximum length: `255`

- **spec.flowLogs.allowFilterList.source_identity**
  array of integers

  Source security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.source_ip**
  array of strings

Source IP or CIDR.

- **spec.flowLogs.allowFilterList.source_ip_xlated**
  array of strings

  Post-translation (SNAT) source IP or CIDR.

- **spec.flowLogs.allowFilterList.source_label**
  array of strings

  Source label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  source_label: k8s:role=frontend
  ```

  ```
  source_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.allowFilterList.source_pod**
  array of strings

  Source Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  source_pod: kube-system/
  ```

  ```
  source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.allowFilterList.source_port**
  array of strings

  L4 source port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.source_service**
  array of strings

  Source service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

  ```
  source_service: kube-system/kube-dns
  ```

  ```
  source_service: kube-system/
  ```

  ```
  source_service: "/kube-dns"
  ```

- **spec.flowLogs.allowFilterList.source_workload**
  array of objects

  Source workloads.

  - **spec.flowLogs.allowFilterList.source_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.allowFilterList.source_workload.name**
    string

    Workload name.

- **spec.flowLogs.allowFilterList.tcp_flags**
  array of objects

  TCP flags to match.

  - **spec.flowLogs.allowFilterList.tcp_flags.ACK**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.CWR**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.ECE**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.FIN**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.NS**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.PSH**

boolean
- **spec.flowLogs.allowFilterList.tcp_flags.RST**
boolean
- **spec.flowLogs.allowFilterList.tcp_flags.SYN**
boolean
- **spec.flowLogs.allowFilterList.tcp_flags.URG**
boolean

- **spec.flowLogs.allowFilterList.trace_id**
array of strings

Trace ID.

- **spec.flowLogs.allowFilterList.traffic_direction**
array of strings

Traffic direction.

- **Element of the array**
string

Allowed values: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`

- **spec.flowLogs.allowFilterList.uuid**
array of strings

Filters by flow UUIDs.

- **spec.flowLogs.allowFilterList.verdict**
array of strings

Match by verdict.

- **Element of the array**
string

Allowed values: `VERDICT_UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`

- **spec.flowLogs.denyFilterList**
array of objects

Defines filters that control which events are explicitly excluded from logging. Any event that matches at least one of the specified filter objects will be denied from being exported. If no filters are specified, no events are denied. This allows you to refine the set of events that are logged by excluding certain types. Logical semantics: - Within a single filter object, all specified fields are combined using AND. - Across filter objects in the array, filters are combined using OR. For more details, see the Cilium documentation.

- **spec.flowLogs.denyFilterList.destination_cluster_name**
array of strings

Destination cluster names.

- **spec.flowLogs.denyFilterList.destination_fqdn**
array of strings

Destination fully qualified domain names (FQDN).

- **Element of the array**
string

Maximum length: `255`

- **spec.flowLogs.denyFilterList.destination_identity**
array of integers

Destination security identity.

- **Element of the array**
integer

Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.destination_ip**
array of strings

Destination IP or CIDR.

- **spec.flowLogs.denyFilterList.destination_label**
array of strings

Destination label selectors.

For more information, please see the Cilium documentation.

Examples:

```
destination_label: k8s:role=frontend
```

```
destination_label: reserved:kube-apiserver
```

- **spec.flowLogs.denyFilterList.destination_pod**
array of strings

Destination Pod name prefixes in the form NamespacedName (`namespace/pod`).

Examples:

```
destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
destination_pod: kube-system/
```

```
destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.denyFilterList.destination_port**
  array of strings

  L4 destination port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.destination_service**
  array of strings

  Destination service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

```
destination_service: kube-system/kube-dns
```

```
destination_service: kube-system/
```

```
destination_service: "/kube-dns"
```

- **spec.flowLogs.denyFilterList.destination_workload**
  array of objects

  Destination workloads.

  - **spec.flowLogs.denyFilterList.destination_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.denyFilterList.destination_workload.name**
    string

    Workload name.

- **spec.flowLogs.denyFilterList.dns_query**
  array of strings

  DNS query regex patterns.

- **spec.flowLogs.denyFilterList.drop_reason_desc**
  array of strings

  Drop reason (only for DROPPED verdict).

  - **Element of the array**
    string

    Allowed values: `DROP_REASON UNKNOWN`, `INVALID SOURCE MAC`, `INVALID_DESTINATION_MAC`, `INVALID_SOURCE IP`, `POLICY_DENIED`, `INVALID_PACKET_DROPPED`, `CT TRUNCATED OR INVALID HEADER`, `CT_MISSING TCP ACK FLAG`, `CT_UNKNOWN L4 PROTOCOL`, `CT_CANNOT_CREATE_ENTRY_FROM_PACKET`, `UNSUPPORTED_L3_PROTOCOL`, `MISSED_TAIL_CALL`, `ERROR_WRITING_TO_PACKET`, `UNKNOWN_L4_PROTOCOL`, `UNKNOWN_ICMPV4_CODE`, `UNKNOWN ICMPV4 TYPE`, `UNKNOWN_ICMPV6 CODE`, `UNKNOWN ICMPV6 TYPE`, `ERROR_RETRIEVING_TUNNEL_KEY`, `ERROR_RETRIEVING_TUNNEL_OPTIONS`, `INVALID_GENEVE_OPTION`, `UNKNOWN_L3_TARGET_ADDRESS`, `STALE_OR_UNROUTABLE IP`, `NO MATCHING LOCAL CONTAINER_FOUND`, `ERROR_WHILE_CORRECTING_L3_CHECKSUM`, `ERROR_WHILE_CORRECTING_L4_CHECKSUM`, `CT_MAP_INSERTION_FAILED`, `INVALID IPV6 EXTENSION HEADER`, `IP_FRAGMENTATION_NOT_SUPPORTED`, `SERVICE_BACKEND_NOT_FOUND`, `NO_TUNNEL_OR_ENCAPSULATION_ENDPOINT`, `FAILED TO INSERT INTO PROXYMAP`, `REACHED_EDT_RATE_LIMITING_DROP_HORIZON`, `UNKNOWN_CONNECTION_TRACKING_STATE`, `LOCAL_HOST_IS_UNREACHABLE`, `NO_CONFIGURATION_AVAILABLE_TO_PERFORM_POLICY_DECISION`, `UNSUPPORTED_L2_PROTOCOL`, `NO_MAPPING_FOR_NAT_MASQUERADE`, `UNSUPPORTED_PROTOCOL_FOR_NAT_MASQUERADE`, `FIB_LOOKUP_FAILED`, `ENCAPSULATION_TRAFFIC_IS_PROHIBITED`, `INVALID_IDENTITY`, `UNKNOWN_SENDER`, `NAT_NOT_NEEDED`, `IS_A_CLUSTERIP`, `FIRST_LOGICAL_DATAGRAM_FRAGMENT_NOT_FOUND`, `FORBIDDEN_ICMPV6_MESSAGE`, `DENIED_BY_LB_SRC_RANGE_CHECK`, `SOCKET_LOOKUP_FAILED`, `SOCKET_ASSIGN_FAILED`, `PROXY_REDIRECTION_NOT_SUPPORTED_FOR_PROTOCOL`, `POLICY_DENY`, `VLAN_FILTERED`, `INVALID_VNI`, `INVALID_TC_BUFFER`, `NO_SID`, `MISSING_SRV6_STATE`, `NAT46`, `NAT64`, `AUTH_REQUIRED`, `CT_NO_MAP_FOUND`, `SNAT_NO_MAP_FOUND`, `INVALID_CLUSTER_ID`, `UNSUPPORTED_PROTOCOL_FOR_DSR_ENCAP`, `NO_EGRESS_GATEWAY`, `UNENCRYPTED_TRAFFIC`, `TTL_EXCEEDED`, `NO_NODE_ID`, `DROP_RATE_LIMITED`, `IGMP_HANDLED`, `IGMP_SUBSCRIBED`, `MULTICAST_HANDLED`, `DROP_HOST_NOT_READY`, `DROP_EP_NOT_READY`, `DROP_NO_EGRESS_IP`

- **spec.flowLogs.denyFilterList.event_type**
  array of objects

Event types to filter on.

- **spec.flowLogs.denyFilterList.event_type.match_sub_type**
  boolean

  Whether to match on `sub_type`.

- **spec.flowLogs.denyFilterList.event_type.sub_type**
  integer

  Secondary event type.

- **spec.flowLogs.denyFilterList.event_type.type**
  integer

  Primary event type.

- **spec.flowLogs.denyFilterList.http_header**
  array of objects

  HTTP headers to match.

  - **spec.flowLogs.denyFilterList.http_header.key**
    string

    Header key.

  - **spec.flowLogs.denyFilterList.http_header.value**
    string

    Header value.

- **spec.flowLogs.denyFilterList.http_method**
  array of strings

  HTTP methods.

  - **Element of the array**
    string

    Allowed values: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`

- **spec.flowLogs.denyFilterList.http_path**
  array of strings

  Regex list to match HTTP path.

- **spec.flowLogs.denyFilterList.http_status_code**
  array of strings

  HTTP status code prefixes.

  - **Element of the array**
    string

    Pattern: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`

- **spec.flowLogs.denyFilterList.http_url**
  array of strings

  Regex list to match HTTP URL.

- **spec.flowLogs.denyFilterList.interface**
  array of objects

  Network interface where flow was observed.

  - **spec.flowLogs.denyFilterList.interface.index**
    integer

    Interface index.

    Allowed values: `0 <= X`

  - **spec.flowLogs.denyFilterList.interface.name**
    string

    Interface name.

- **spec.flowLogs.denyFilterList.ip_version**
  array of strings

  IP version.

  - **Element of the array**
    string

    Allowed values: `IP_NOT_USED`, `IPv4`, `IPv6`

- **spec.flowLogs.denyFilterList.node_labels**
  array of strings

  Node label selectors.

  Examples:

```
node_labels: node-role.kubernetes.io/control-plane=
```

```
node_labels: topology.kubernetes.io/ru-central1-a
```

- **spec.flowLogs.denyFilterList.node_name**
  array of strings

  Node name patterns.

- **spec.flowLogs.denyFilterList.protocol**
  array of strings

  L4 or L7 protocol.

- **spec.flowLogs.denyFilterList.source_cluster_name**
  array of strings

  Source cluster names.

- **spec.flowLogs.denyFilterList.source_fqdn**
  array of strings

  Source fully qualified domain names (FQDN).

  - **Element of the array**
    string

    Maximum length: `255`

- **spec.flowLogs.denyFilterList.source_identity**
  array of integers

  Source security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.source_ip**
  array of strings

  Source IP or CIDR.

- **spec.flowLogs.denyFilterList.source_ip_xlated**
  array of strings

  Post-translation (SNAT) source IP or CIDR.

- **spec.flowLogs.denyFilterList.source_label**
  array of strings

  Source label selectors.

  For more information, please see the Cilium documentation.

  Examples:

```
source_label: k8s:role=frontend
```

```
source_label: reserved:kube-apiserver
```

- **spec.flowLogs.denyFilterList.source_pod**
  array of strings

  Source Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

```
source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
source_pod: kube-system/
```

```
source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.denyFilterList.source_port**
  array of strings

  L4 source port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.source_service**
  array of strings

  Source service name prefixes in the form NamespacedName (`namespace/service`).

Examples:

```
source_service: kube-system/kube-dns
```

```
source_service: kube-system/
```

```
source_service: "/kube-dns"
```

- **spec.flowLogs.denyFilterList.source_workload**
  array of objects

  Source workloads.

  - **spec.flowLogs.denyFilterList.source_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.denyFilterList.source_workload.name**
    string

    Workload name.

- **spec.flowLogs.denyFilterList.tcp_flags**
  array of objects

  TCP flags to match.

  - **spec.flowLogs.denyFilterList.tcp_flags.ACK**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.CWR**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.ECE**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.FIN**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.NS**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.PSH**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.RST**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.SYN**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.URG**
    boolean
- **spec.flowLogs.denyFilterList.trace_id**
  array of strings

  Trace ID.

- **spec.flowLogs.denyFilterList.traffic_direction**
  array of strings

  Traffic direction.

  - **Element of the array**
    string

    Allowed values: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`

- **spec.flowLogs.denyFilterList.uuid**
  array of strings

  Filters by flow UUIDs.

- **spec.flowLogs.denyFilterList.verdict**
  array of strings

  Match by verdict.

  - **Element of the array**
    string

    Allowed values: `VERDICT_UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`

- **spec.flowLogs.enabled**
  boolean

  Enables event logging for Cilium agents.

  > **i** The log file is located on the node's filesystem at `/var/log/cilium/hubble/flow.log`.

  Default: `false`

- **spec.flowLogs.fieldMaskList**
  array of strings

List of fields that should be displayed in events.

For more details, see the Cilium documentation.

- **Element of the array**
  string

  Allowed values: `time`, `uuid`, `verdict`, `drop_reason_desc`, `auth_type`, `ethernet`, `ethernet.source`, `ethernet.destination`, `IP`, `IP.source`, `IP.source_xlated`, `IP.destination`, `IP.ipVersion`, `IP.encrypted`, `l4`, `l4.TCP`, `l4.TCP.source_port`, `l4.TCP.destination_port`, `l4.TCP.flags`, `l4.TCP.flags.FIN`, `l4.TCP.flags.SYN`, `l4.TCP.flags.RST`, `l4.TCP.flags.PSH`, `l4.TCP.flags.ACK`, `l4.TCP.flags.URG`, `l4.TCP.flags.ECE`, `l4.TCP.flags.CWR`, `l4.TCP.flags.NS`, `l4.UDP`, `l4.UDP.source port`, `l4.UDP.destination port`, `l4.ICMPv4`, `l4.ICMPv4.type`, `l4.ICMPv4.code`, `l4.ICMPv6`, `l4.ICMPv6.type`, `l4.ICMPv6.code`, `l4.SCTP`, `l4.SCTP.source port`, `l4.SCTP.destination port`, `source`, `source.ID`, `source.identity`, `source.cluster_name`, `source.namespace`, `source.labels`, `source.pod_name`, `source.workloads`, `source.workloads.name`, `source.workloads.kind`, `destination`, `destination.ID`, `destination.identity`, `destination.cluster_name`, `destination.namespace`, `destination.labels`, `destination.pod_name`, `destination.workloads`, `destination.workloads.name`, `destination.workloads.kind`, `Type`, `node_name`, `node_labels`, `source_names`, `destination_names`, `l7`, `l7.type`, `l7.latency_ns`, `l7.record`, `l7.record.dns`, `l7.record.dns.query`, `l7.record.dns.ips`, `l7.record.dns.ttl`, `l7.record.dns.cnames`, `l7.record.dns.observation_source`, `l7.record.dns.rcode`, `l7.record.dns.qtypes`, `l7.record.dns.rrtypes`, `l7.record.http`, `l7.record.http.code`, `l7.record.http.method`, `l7.record.http.url`, `l7.record.http.protocol`, `l7.record.http.headers`, `l7.record.http.headers.key`, `l7.record.http.headers.value`, `l7.record.kafka`, `l7.record.kafka.error_code`, `l7.record.kafka.api_version`, `l7.record.kafka.api_key`, `l7.record.kafka.correlation_id`, `l7.record.kafka.topic`, `is_reply`, `event_type`, `event_type.type`, `event_type.sub_type`, `traffic_direction`, `trace_observation_point`, `trace_reason`, `file`, `file.name`, `file.line`, `debug_capture_point`, `interface`, `interface.index`, `interface.name`, `proxy_port`, `trace_context`, `trace_context.parent`, `trace_context.parent.trace_id`, `sock_xlate_point`, `socket_cookie`, `cgroup_id`, `extensions`, `egress_allowed_by`, `egress_allowed_by.name`, `egress_allowed_by.namespace`, `egress_allowed_by.labels`, `egress_allowed_by.revision`, `egress_allowed_by.kind`, `ingress_allowed_by`, `ingress_allowed_by.name`, `ingress_allowed_by.namespace`, `ingress_allowed_by.labels`, `ingress_allowed_by.revision`, `ingress_allowed_by.kind`, `egress_denied_by`, `egress_denied_by.name`, `egress_denied_by.namespace`, `egress_denied_by.labels`, `egress_denied_by.revision`, `egress_denied_by.kind`, `ingress_denied_by`, `ingress_denied_by.name`, `ingress_denied_by.namespace`, `ingress_denied_by.labels`, `ingress_denied_by.revision`, `ingress_denied_by.kind`, `source_service`, `source_service.name`, `source_service.namespace`, `destination_service`, `destination_service.name`, `destination_service.namespace`

- **spec.flowLogs.fileMaxSizeMB**
  integer

  Size in MB at which to rotate the Hubble export file.

  Default: `10`

  Allowed values: `1 <= X <= 150`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> The module lifecycle stage: General Availability

Cluster peering

When the administrator has meshed the cluster with other clusters (see Cluster peering in advanced usage), a pod reaches pods of other clusters by IP address, and a service can use backends of other clusters.

Pods of other clusters

A pod reaches a pod of another cluster by its IP address, like a pod of its own cluster. There is no NAT: the receiving pod sees the client pod's address.

Pods and services of other clusters have no DNS names. To reach an application of another cluster by name, make its service global.

Global services

A global service uses its own backends and the backends of the service with the same name in the same namespace in the neighbour clusters. Clients reach it by its name and address in their own cluster.

A service is configured with annotations:

Annotation	Value

Annotation	Value
`network.deckhouse.io/global-service-mode`	`Local`: the service is local (same as no annotation). `ImportOnly`: uses the neighbours' backends, does not offer its own. `ImportAndExport`: uses the neighbours' backends and offers its own
`network.deckhouse.io/global-service-traffic-distribution`	`Any` (default): backends of all clusters. `PreferLocalCluster`: own backends while any is ready. `PreferRemoteCluster`: the neighbours' backends while any is ready

An unknown mode means `Local`, an unknown distribution means `Any`. The distribution applies only in the `ImportOnly` and `ImportAndExport` modes. Changes apply without restarting pods; open connections are not moved.

The service name, namespace and ports (with names) must match in every cluster.

A headless service (`clusterIP: None`) can be global too: its DNS name returns the addresses of the pods of this and the neighbour clusters. The traffic distribution does not apply to it. Name the StatefulSet pods behind such a service differently in each cluster, for example with the cluster name: equal pod names from different clusters conflict in DNS.

Common tasks:

Task	What to do
Balance over the replicas of all clusters	`global-service-mode: ImportAndExport` in every cluster
Use own replicas, the neighbours as a fallback	Add `global-service-traffic-distribution: PreferLocalCluster`
Use an application that does not run in your cluster	In your cluster, a service with `ImportOnly`; in the cluster running the application, with `ImportAndExport`
Drain a cluster without stopping its replicas	In that cluster, `ImportOnly` and `PreferRemoteCluster`. After maintenance, set `ImportAndExport` back and remove the distribution

Example:

```
apiVersion: v1
kind: Service
metadata:
  name: api
  namespace: shop
  annotations:
    network.deckhouse.io/global-service-mode: ImportAndExport
    network.deckhouse.io/global-service-traffic-distribution: PreferLocalCluster
spec:
  selector:
    app: api
  ports:
  - name: http
    port: 80
    targetPort: 8080
```

When the mesh with a neighbour is torn down, its backends are removed from global services and connections to its pods break. A temporary loss of connectivity between the clusters does not tear the mesh down.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Advanced usage

> The module lifecycle stage: General Availability

Cluster peering

The module connects the pod networks of clusters peered by the `peering` module: a pod reaches a pod of another cluster by IP address, a service can use backends of other clusters (see Cluster peering in usage). The clusters only exchange the mesh settings over peering; traffic goes directly between the nodes.

Limitations:

- VXLAN mode without encryption only; pod traffic between clusters is not encrypted, so mesh clusters over trusted networks only;
- neighbours connect to control-plane nodes directly, by node addresses; a load balancer or NAT between clusters is not supported;
- the pod subnets of the clusters must not overlap;
- a cluster with all its neighbours is at most 255 clusters;

- the mesh is not transitive: if A is meshed with B and B with C, A and C are not connected;
- enabling, disabling and changing the cluster ID change the security identities of all pods: restart all pods of the cluster afterwards, in a maintenance window.

Requirements

Requirement	How to check
The `peering` module is enabled, `ClusterPeer` is `Active` on both sides	`d8 k get clusterpeers`
Cilium runs in VXLAN mode, `tunnelMode: VXLAN` is set explicitly in the ModuleConfig	`d8 k get mc cni-cilium -o jsonpath='{.spec.settings.tunnelMode}'` prints `VXLAN`
Encryption is off: `encryption.mode` is unset or `Disabled`	`d8 k get mc cni-cilium -o jsonpath='{.spec.settings.encryption.mode}'`
The same VXLAN port in all clusters	`d8 k -n d8-cni-cilium get cm cilium-config -o jsonpath='{.data.tunnel-port}'`
The pod subnet does not overlap the neighbours' pod subnets	`podSubnetCIDR` in `d8 k -n kube-system get secret d8-cluster-configuration -o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d`
The cluster ID is unique among the cluster and all its neighbours	Assign IDs from one list for all clusters that may be meshed

The module sets the VXLAN port. Clusters with different ports cannot be meshed. Enabling the `virtualization` module can change the port.

Service subnets may overlap.

Network access between clusters, by node `InternalIP` addresses, without NAT:

From	To	Port
Every node	Every node of the neighbour	UDP, the VXLAN port
Control-plane nodes	Control-plane nodes of the neighbour	TCP, the neighbour's `networkPeering.hostPort.port` (4294 by default)

Enabling

The example meshes clusters A and B. The peering is named `cluster-b` in cluster A and `cluster-a` in cluster B. Run every step in both clusters.

1. Peer the clusters by the `peering` module guide up to the `Active` phase. Do not open capabilities yet.
2. Enable NetworkPeering in the `cni-cilium` ModuleConfig (`d8 k edit mc cni-cilium`). In cluster B, `clusterID: 2`:

   ```
   spec:
     settings:
       tunnelMode: VXLAN
       networkPeering:
         enabled: true
         clusterID: 1
   ```

   The Cilium agents restart. Then restart all pods of the cluster so they get identities with the cluster ID. Do it in a maintenance window. If Cilium runs without a tunnel, switch it to VXLAN in a separate maintenance window first.

   Wait for `READY: True` in `d8 k get networkpeering cluster` and `ACCEPTED: True` in `d8 k get peeringcapability network`.
3. Add `network` to `spec.capabilities` of the `ClusterPeer`, keeping the other capabilities (`d8 k edit clusterpeer cluster-b`). The module creates a ClusterNetworkPeer named after the `ClusterPeer`, it shows the state of the mesh with the neighbour. The capability works once both sides list it: check `EFFECTIVE` in `d8 k get clusterpeer cluster-b`.
4. Wait for `PHASE: Configured` and `READY: True` in `d8 k get clusternetworkpeers`. It usually takes a minute or two.

If the ModuleConfig is rejected, fix the setting named in the message: set `networkPeering.clusterID`, an explicit `tunnelMode: VXLAN`, or turn encryption off.

Several neighbours

The cluster meshes with each neighbour separately: the `ClusterPeer` lists the `network` capability on both sides. The module keeps a ClusterNetworkPeer for every `ClusterPeer` that lists `network` and deletes it when the capability or the `ClusterPeer` is removed.

A neighbour's ID and pod subnet stay taken while the cluster has its ClusterNetworkPeer, even if the mesh with it is down. A new neighbour with a taken ID or subnet gets an error (`ClusterIDConflict`, `PodCIDROverlap`); the neighbour already meshed keeps working. To resolve the conflict, change the new neighbour's ID or remove `network` from the old neighbour's `ClusterPeer`.

ClusterNetworkPeer status

Phases:

- `Pending`: the mesh waits for the neighbour or for the peering, the reason is in the conditions;
- `Negotiating`: certificates are being exchanged;
- `Configured`: the mesh is configured;
- `Error`: a check failed, the reason is in the conditions.

To print the conditions, run:

```
d8 k get clusternetworkpeer cluster-b \
  -o jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}
{end}'
```

Reason	What to do
`PeerNotActive`	Check the peering
`CapabilityNotOffered`	On the neighbour's side, add `network` to the `ClusterPeer` and enable NetworkPeering
`CapabilityNotAccepted`	This cluster does not offer `network` to the neighbour: check the `Accepted` and `GrantsApplied` conditions in `d8 k get peeringcapability network -o yaml`
`CapabilityNotEffective`	Both sides offer `network`, the peering has not applied it yet. If it persists, check `d8 k get clusterpeer cluster-b`
`AwaitingPeer`	The neighbour has not created its ClusterNetworkPeer named in the message yet. If it persists, check `d8 k -n d8-cni-cilium logs deploy/clustermesh-controller` on the neighbour
`PeerUnreachable`, `ClusterPeerUnreadable`, `PeerAccessNotReady`	Check the peering. With `... not found`, wait until NetworkPeering is deployed on the neighbour
`TunnelModeNotVXLAN`, `VXLANPortMismatch`	Switch the neighbour to VXLAN with the same port. Clusters with different ports cannot be meshed
`ClusterIDConflict`	Change the neighbour's ID or remove `network` from the `ClusterPeer` of the neighbour named in the message
`ClusterNameConflict`	Delete the extra peering with the same cluster. A cluster cloned with its UUID cannot be meshed
`PodCIDROverlap`	The pod subnets (podCIDR) of the clusters overlap. These clusters cannot be meshed
`RemoteNotReady`	The neighbour has not published its state yet. If it persists, check `d8 k get networkpeering cluster` and the `clustermesh-apiserver` replicas on the neighbour
`AwaitingPeerSignature`	Wait for the next check. If it persists, look at the neighbour's ClusterNetworkPeer
`PeerCSRInvalid`	On the neighbour's side, remove `network` from the `ClusterPeer`, wait until its ClusterNetworkPeer is deleted, and add `network` back
`LastKnownConfiguration`	The peering is unavailable, the mesh runs on the last configuration. Restore the peering

Changing settings

- To change `networkPeering.clusterID`, check that the new ID is free among the cluster and its neighbours. Change it in a maintenance window, then restart all pods of the cluster.
- To change `networkPeering.hostPort.port`, open the new port in the neighbours' firewalls first.

Disconnecting a neighbour

To disconnect a neighbour, remove `network` from `spec.capabilities` of its `ClusterPeer` on either side (`d8 k edit clusterpeer cluster-b`). The module deletes the ClusterNetworkPeer.

The mesh is torn down on both sides, connections between the clusters break. To restore the mesh, add `network` back.

To give the ID or pod subnet of a retired cluster to another cluster, remove `network` from the retired cluster's `ClusterPeer` on all its neighbours, or delete that `ClusterPeer`.

Disabling NetworkPeering

1. Remove `network` from `spec.capabilities` of the `ClusterPeer` objects. Wait until `d8 k get clusternetworkpeers` lists none: the module deletes them within a minute.
2. Set `networkPeering.enabled: false` in the `cni-cilium` ModuleConfig in a maintenance window, then restart all pods of the cluster.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Egress Gateway

Operation principle

Configuring an egress gateway requires two custom resources:

1. EgressGateway— describes the group of nodes that perform the egress gateway function in hot-standby mode:
   - Among the group of nodes match the `spec.nodeSelector`, the eligible nodes will be selected. One of them will be assigned as the active gateway. The active node is selected in alphabetical order.

     Attributes of an eligible node:

     - The node is in `Ready` state.
       - The node is not in the maintenance state (i.e., it is not cordoned).
       - The `cilium-agent` on the node is in the `Ready` state.
     - When using EgressGateway in `VirtualIP` mode, an agent is launched on the active node which emulates a “virtual” IP address using the ARP protocol. The status of this agent’s pod is also taken into account when determining the eligibility of a node.
     - Different EgressGateways can use the same nodes for operation. The active node is selected independently for each EgressGateway, which allows for load balancing between them.
2. EgressGatewayPolicy — describes the policy for routing network requests from pods in the cluster to a specific egress gateway defined using EgressGateway.

Node maintenance

To perform maintenance on a node that is currently the active egress gateway, follow these steps:

1. Remove the node label to exclude it from the egress gateway candidate pool. Egress-label is the label specified in `spec.nodeSelector` of your EgressGateway.

   ```
   d8 k label node <node-name> <egress-label>-
   ```

2. Cordon the node to prevent new pods from starting:

   ```
   d8 k cordon <node-name>
   ```

   After this, Cilium will automatically select a new active node from the remaining candidates. Traffic will continue routing through the new gateway without interruption.

3. After maintenance is complete, return the node to service:

   ```
   d8 k uncordon <node-name>
   d8 k label node <node-name> <egress-label>=<value>
   ```

> Reapplying the label may cause the node to become active again (if it is first in alphabetical order among candidates).

To avoid immediate failback, temporarily reduce the number of EgressGateway replicas or adjust priorities using additional labels.

Comparison with CiliumEgressGatewayPolicy

The CiliumEgressGatewayPolicy implies configuring only one node as an egress gateway. If it fails, there are no failover mechanisms and the network connection will be broken.

Egress Gateway configuration examples

EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # The "public" interface must have the same name on all nodes that matching the nodeSelector.
      # If the active node fails, traffic will be redirected through the backup node and
      # the source IP address of the network packets will change.
      interfaceName: eth1
```

EgressGateway in VirtualIPAddress mode (Virtual IP mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # Each node must have all the necessary routes configured to access all external public services,
      # the "public" interface must be prepared for automatic configuration of the "virtual" IP as a
secondary IP address.
      # In case of failure of the active node, traffic will be redirected through the backup node and
      # the source IP address of the network packets will not change.
      ip: 172.18.18.242
      # List of network interfaces for Virtual IP
      interfaces:
      - eth1
```

EgressGatewayPolicy

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
  - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
  - podSelector:
      matchLabels:
        app: backend
        io.kubernetes.pod.namespace: my-ns
```

HubbleMonitoringConfig

The cluster-scoped HubbleMonitoringConfig resource is intended to configure data export from Hubble, which runs inside Cilium agents.

HubbleMonitoringConfig configuration examples

Enabling extended metrics and flow logs export (with filters and field mask)

> ⚠ The HubbleMonitoringConfig resource must be named `hubble-monitoring-config`.

Example of enabling metrics and export:

```
apiVersion: deckhouse.io/v1alpha1
kind: HubbleMonitoringConfig
metadata:
  name: hubble-monitoring-config
spec:
  extendedMetrics:
    enabled: true
    collectors:
      - name: drop
        # Add additional labels context for the selected collector.
        contextOptions:
"labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,destination_po
d"
      - name: flow
  flowLogs:
    enabled: true
    # Allow only the specified events to be written to the log file /var/log/cilium/hubble/flow.log.
    allowFilterList:
      - verdict: ["DROPPED", "ERROR"]
    # Exclude events matching the denyFilterList from the log file.
    denyFilterList:
      - source_pod:
        - kube-system/
      - destination_pod:
        - kube-system/
    # Persist only the specified fields in each record.
    fieldMaskList:
      - time
      - verdict
    # Maximum log file size (in MB) before rotation.
    fileMaxSizeMB: 30
```

Collecting Hubble flow logs with the log-shipper module

To collect flow logs, use the `log-shipper` module.

Create a ClusterLoggingConfig resource that reads the log file from the node filesystem:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: cilium-hubble-flow-logs
spec:
  type: File
  file:
    include:
      - /var/log/cilium/hubble/flow.log
```

Enabling BPF trace events on demand

BPF trace events (`bpf-events-trace-enabled`) are disabled cluster-wide by default. They are the dominant source of `cilium-agent` CPU and memory on nodes with high pod density: each forwarded packet (subject to `monitor-aggregation`) produces an entry in the BPF event buffer that the agent has to parse, label and feed into Hubble. `drop` and `policy verdict` events are not affected by this setting and remain available in Hubble.

When forwarded-flow visibility is required (for example, to debug a connectivity issue), enable trace events with a CiliumNodeConfig resource. The same resource can target a single node, a group of nodes, or all nodes in the cluster — depending on `spec.nodeSelector`.

Enable trace events on a single node

To enable trace on a specific node, specify its name in the `spec.nodeSelector.matchLabels` parameter:

```
apiVersion: cilium.io/v2
kind: CiliumNodeConfig
metadata:
  name: trace-debug-node-1
  namespace: d8-cni-cilium
spec:
  nodeSelector:
    matchLabels:
      kubernetes.io/hostname: <node-name>
  defaults:
    bpf-events-trace-enabled: "true"
    # Aggregation level "none" disables aggregation of trace events for packets of the
    # same session: a separate trace event is created for each packet.
    # If this line is omitted, the cluster default "medium" remains on the node.
    monitor-aggregation: "none"
```

Enable trace events cluster-wide

An empty `matchLabels` selects all nodes:

```
apiVersion: cilium.io/v2
kind: CiliumNodeConfig
metadata:
  name: trace-debug-all-nodes
  namespace: d8-cni-cilium
spec:
  nodeSelector:
    matchLabels: {}
  defaults:
    bpf-events-trace-enabled: "true"
```

Applying the change

Cilium agent at startup reads the merged configuration, so the affected `cilium-agent` pods must be restarted after the CiliumNodeConfig is created or modified (Deckhouse does not restart agents automatically on such changes):

- for single node:

  ```
  d8 k -n d8-cni-cilium delete pod \
    -l app=agent --field-selector spec.nodeName=<node-name>
  ```

- for all nodes (rolling restart):

  ```
  d8 k -n d8-cni-cilium rollout restart daemonset/agent
  ```

To revert (disable traces), delete the CiliumNodeConfig resource and restart the same pods.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The cni-flannel module

> The module lifecycle stage: General Availability

This module is responsible for providing a network between multiple nodes in a cluster using the flannel module.

Third-party components

List of third-party software used in the `cni-flannel` module:

- Flannel 0.27.4

  License: Apache License 2.0

  A simple and easy way to configure a layer 3 network fabric designed for Kubernetes.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

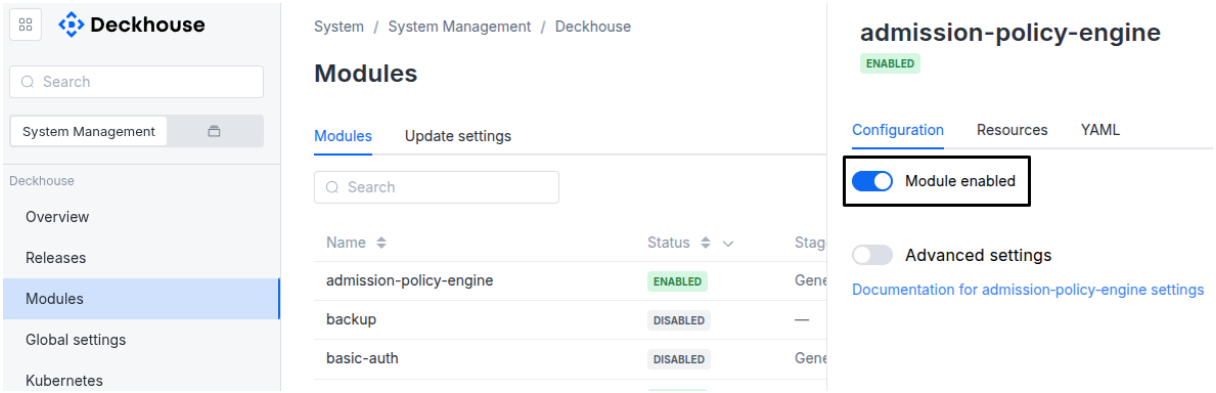
The module has 7 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-flannel` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cni-flannel
```

- **Using ModuleConfig `cni-flannel`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cni-flannel` (create it if necessary);

  Example of a manifest to enable module `cni-flannel`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-flannel
spec:
  enabled: true
```

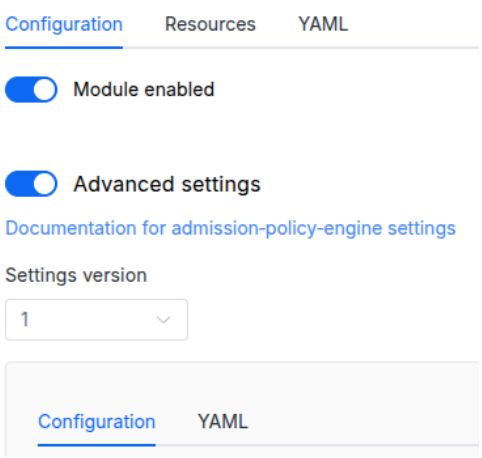
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-flannel` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cni-flannel` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cni-flannel`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cni-flannel` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cni-flannel
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cni-flannel` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cni-flannel`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cni-flannel
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.ipv4OptionsGuard**
    object

    Mitigation of CVE-2026-98432 (use-after-free in the Linux IPv4 TCP handshake triggered by a TCP SYN with IP options).

    A sidecar of the CNI agent installs a filter into the host network namespace and into every pod network namespace on the node, including pods started later. The filter matches IPv4 TCP SYN packets and IPv4 fragments carrying IP options.

    The filter follows the iptables mode (nft or legacy) that iptables-wrapper selects on the node, the same as the other Deckhouse components. In the nft mode, the filter uses native nftables expressions. In the legacy mode, it uses iptables with the `u32` match. No kernel modules are loaded unless `allowKernelModuleLoad` is set. If the kernel modules of the selected mode are not loaded but those of the other mode are, the filter uses the other mode, and the node is reported by the `D8IPv4OptionsGuardBackendMismatch` alert. A node that cannot be protected is reported by the `D8IPv4OptionsGuardNodeUnprotected` alert.

    To exclude a node (for example, a node with CIPSO labeled networking), set the `ipv4-options-guard.deckhouse.io/disabled: "true"` label on it: the changes are reverted on that node.

    The mitigation does not fix the vulnerability: update the node kernel as soon as a fixed version is available.

    - **settings.ipv4OptionsGuard.allowKernelModuleLoad**
      boolean

      The filter needs the `nf_tables` kernel module in the nft mode and the `ip_tables`, `iptable_raw`, `xt_tcpudp` and `xt_u32` kernel modules in the legacy mode.

      By default, kernel modules are not loaded: if a module of the mode selected by iptables-wrapper is not loaded, the filter uses the other mode when its modules are loaded, and the node is reported as unprotected otherwise. Set to `true` to load the modules of the selected mode.

      Default: `false`

    - **settings.ipv4OptionsGuard.disableMPTCP**
      boolean

      In the `Enforce` mode, set `net.mptcp.enabled=0` in the host and in every pod network namespace. This removes the precondition of the published exploit.

      Listeners opened before the change stay MPTCP until the process is restarted. After a node reboot, MPTCP on the host is disabled again once the CNI agent starts.

      Default: `true`

    - **settings.ipv4OptionsGuard.mode**
      string

      Operating mode.

If the mode is not set, it is `Enforce` in clusters installed with the mitigation and `Audit` in clusters that existed before it.

- `Audit` — matching packets are only counted (the `d8_ipv4_options_guard_matched_packets_total` metric and the `D8IPv4OptionsGuardMatchedTraffic` alert). Use it to check that no workload relies on IP options. While any node is in this mode, the `D8IPv4OptionsGuardAuditMode` alert reminds that the cluster is not protected yet.
- `Enforce` — matching packets are dropped, and MPTCP is disabled if `disableMPTCP` is set.
- `Disabled` — the filter is removed from all network namespaces, including running pods, and `net.mptcp.enabled` is restored to the value it had before.

The mode is applied without restarting the CNI agent pods.

Allowed values: `Audit`, `Enforce`, `Disabled`

- **settings.podNetworkMode**
  string

  Work mode.

  > **NOTE!** All cluster nodes must be rebooted after switching work mode. You can change the parameter **only** in bare metal clusters.

  Default: `HostGW`

  Allowed values: `HostGW`, `VXLAN`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The cni-simple-bridge module

> The module lifecycle stage: General Availability

The module does not have any settings.

It is automatically enabled for the following cloud providers:

- AWS
- Azure
- GCP

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing control plane

> The module lifecycle stage: General Availability

The `control-plane-manager` (CPM) module is responsible for managing the cluster's control plane components. It runs on all master nodes of the cluster (nodes that have the `node-role.kubernetes.io/control-plane: ""` label).

The control-plane-manager:

- **Manages certificates** required for the operation of the control plane (renews certificates and re-issues them in response to configuration changes, among other things). This feature allows the CPM to automatically maintain a secure control plane configuration and quickly add additional SANs for organizing secure access to the Kubernetes API.
- **Configures components**. The CPM module automatically creates the required configs and manifests of the control plane components.
- **Upgrades/downgrades components**. Makes sure that the versions of the components in the cluster are the same.
- **Manages the configuration of the etcd cluster** and its members. The CPM module scales master nodes and migrates the cluster from single-master to multi-master (and vice versa).
- **Configures kubeconfig**. The CPM module maintains up-to-date kubeconfig files on control-plane nodes. It generates, renews, and updates kubeconfigs for control-plane components and the admin kubeconfig (`admin.conf`). By default, it creates a symlink for the root user (`/root/.kube/config` -> `admin.conf`). When the user-authz module is enabled, the symlink can be turned off via the `rootKubeconfigSymlink` parameter in the **control-plane-manager** module (see FAQ). The CPM also hardens file permissions on `admin.conf` and `super-admin.conf`.
- **Extends scheduler functionality** by integrating external plugins via webhooks. Manages by KubeSchedulerWebhookConfiguration resource. Allows more complex logic to be used in workload scheduling tasks within the cluster. For example:
  - placing data storage application pods closer to the data itself,
  - state-based node prioritization (network load, storage subsystem status, etc.),
  - dividing nodes into zones, etc.
- **Monitors the control plane**. The CPM module organizes secure metrics collection and provides a basic set of monitoring rules for the control plane components. For details, refer to Control plane monitoring.
- **Periodic etcd defragmentation**. In clusters with three or more etcd members, this feature is enabled by default. For

more details, see the description of the `etcd.defrag` parameter.

Managing certificates

The CPM module manages certificates of the `control-plane` components, such as:

- Server certificates for `kube-apiserver` & `etcd`. These are stored in the secret `d8-pki` of the namespace `kube-system`:
  - the root CA kubernetes certificate (`ca.crt` & `ca.key`),
  - the root CA etcd certificate (`etcd/ca.crt` & `etcd/ca.key`),
  - the RSA certificate and the key for signing Service Accounts (`sa.pub` & `sa.key`),
  - the root CA certificate for the extension API servers (`front-proxy-ca.key` & `front-proxy-ca.crt`).
- Client certificates for connecting `control-plane` components to each other. The CPM module issues, renews, and re-issues if something has changed (e.g., the SAN list). These certificates are stored on the nodes only:
  - The server-side API server certificate (`apiserver.crt` & `apiserver.key`).
  - The client-side certificate for connecting `kube-apiserver` to `kubelet` (`apiserver-kubelet-client.crt` & `apiserver-kubelet-client.key`).
  - The client-side certificate for connecting `kube-apiserver` to `etcd` (`apiserver-etcd-client.crt` & `apiserver-etcd-client.key`).
  - The client-side certificate for connecting `kube-apiserver` to the extension API servers (`front-proxy-client.crt` & `front-proxy-client.key`).
  - The server-side `etcd` certificate (`etcd/server.crt` & `etcd/server.key`).
  - The client-side certificate for connecting `etcd` to other cluster members (`etcd/peer.crt` & `etcd/peer.key`).
  - The client-side certificate for connecting `kubelet` to `etcd` for performing health-checks (`etcd/healthcheck-client.crt` & `etcd/healthcheck-client.key`).

Also, the CPM module lets you add the additional SANs to certificates (this way, you can quickly and effortlessly add more “entry points” to the Kubernetes API).

The CPM module automatically updates the kubeconfig configuration when certificates are changed.

Scaling

The CPM module supports `control plane` running in a *single-master* or *multi-master* mode.

In the *single-master* mode:

- `kube-apiserver` only uses the `etcd` instance that is hosted on the same node;
- A proxy server is configured on the node that responds to localhost, `kube-apiserver` responds to the IP address of the master node.

In the *multi-master* mode, `control plane` components are automatically deployed in a fault-tolerant manner:

- `kube-apiserver` is configured to work with all etcd instances;
- The additional proxy server that processes localhost requests is set up on each master node. By default, the proxy server sends requests to the local `kube-apiserver` instance. If it is unavailable, the proxy tries to connect to other `kube-apiserver` instances.

Scaling master nodes

The `control-plane` nodes are scaled automatically using the `node-role.kubernetes.io/control-plane=""` label:

- Attaching the `node-role.kubernetes.io/control-plane=""` label to a node results in deploying `control plane` components on this node, connecting the new `etcd` node to the etcd cluster, and regenerating all the necessary certificates and config files.
- Removing the `node-role.kubernetes.io/control-plane=""` label results in deleting all `control plane` components on a node, gracefully removing it from the etcd cluster, and regenerating all the necessary config files and certificates.

⚠ Manual `etcd` actions are required when decreasing the number of nodes from two to one. In all other cases, all the necessary actions are performed automatically. Please note that when scaling from any number of master nodes to one, sooner or later at the last step, the situation of scaling nodes from two to one will arise.

Dynamic terminated pod garbage collection threshold

Automatically configures the optimal `--terminated-pod-gc-threshold` based on cluster size:

- **Small clusters** (less than 100 nodes): 1000 terminated Pods.
- **Medium clusters** (from 100 to 300 nodes): 3000 terminated Pods.
- **Large clusters** (from 300 nodes): 6000 terminated Pods.

i Note. This feature only takes effect in environments where the `--terminated-pod-gc-threshold` parameter is configurable. On managed Kubernetes services (such as EKS, GKE, AKS), this setting is controlled by managed provider.

Configuring Control Plane Resource Requests

The module allows you to configure the total CPU and memory resource requests for control plane components on each master node: `kube-apiserver`, `etcd`, `kube-controller-manager`, and `kube-scheduler`.

Use the `resourcesRequests` parameter in the `control-plane-manager` ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    resourcesRequests:
      cpu: 1000m
      memory: 500Mi
```

The specified values are used as a common requests budget for control plane components on each master node. Deckhouse Platform (DP) distributes this budget between control plane static pods when rendering their manifests in the following fixed proportions:

Component	Share	Requests for `cpu: 1000m`	Requests for `memory: 500Mi`
`kube-apiserver`	45%	`450m`	`225Mi`
`etcd`	35%	`350m`	`175Mi`
`kube-controller-manager`	10%	`100m`	`50Mi`
`kube-scheduler`	10%	`100m`	`50Mi`

If CPU or memory requests are not explicitly specified, the module automatically calculates requests for the corresponding resource for each control plane component once a day based on the average usage over the previous seven days, plus a 10% margin so that actual usage stays below the request. In this mode the proportions above do not apply: each component gets a request derived from its own measured usage. Explicitly specifying the total amount of CPU or memory disables automatic request calculation for that resource for all control plane components.

i These settings do not apply if the cluster control plane is managed by a cloud provider, for example in GKE, AKS, or EKS.

Version control

Patch versions of control plane components (i.e. within the minor version, for example, from `1.31.13` to `1.31.14`) are upgraded automatically together with the DP version updates. You can't manage patch version upgrades.

Upgrading **minor versions** of control plane components (e.g. from `1.33.*` to `1.34.*`) can be managed using the `kubernetesVersion` parameter of the `control-plane-manager` ModuleConfig. It specifies tracking the DP default (if set to `Default`) or the desired minor version of the control plane. The default control plane version (to use with `kubernetesVersion: Default`) as well as a list of supported Kubernetes versions can be found in the "Supported Kubernetes and OS versions" section.

The cluster Kubernetes version is resolved in one order: `kubernetesVersion` in the `control-plane-manager` ModuleConfig, then the deprecated `ClusterConfiguration.kubernetesVersion`, then the version the current Deckhouse release defaults to. The ModuleConfig setting wins whenever it is set, `Default` included; while it is unset, the deprecated field still decides. Deckhouse raises the `D8ObsoleteKubernetesVersionFieldInClusterConfiguration` alert for as long as the deprecated field is present at all — including when ModuleConfig already owns the version — and it clears once the field is removed from `ClusterConfiguration`.

Example of pinning a Kubernetes version:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    kubernetesVersion: "1.33"
```

The control plane upgrade is performed in a safe way for both single-master and multi-master clusters. The API server may be temporarily unavailable during the upgrade. At the same time, it does not affect the operation of applications in the cluster and can be performed without scheduling a maintenance window.

If the target version (set with the kubernetesVersion parameter) does not match the current control plane version in the cluster, a smart strategy for changing component versions is applied:

- General remarks
  - Updating in different NodeGroups is performed in parallel. Within each NodeGroup, nodes are updated sequentially, one at a time.
- When upgrading:
  - Upgrades are carried out sequentially, one minor version at a time: 1.33 -> 1.34, 1.34 -> 1.35, 1.35 -> 1.36, 1.36 -> 1.37.
  - At each step, the control plane version is upgraded first, followed by kubelet upgrades on the cluster nodes.
- When downgrading:
  - Successful downgrading is only guaranteed for a single version down from the maximum minor version of the control plane ever used in the cluster.
  - kubelets on the cluster nodes are downgraded first, followed by the control plane components.

Exposing the Kubernetes API

The kube-apiserver component is only available within the internal cluster network by default. This module solves the problem of providing simple and secure external access to the Kubernetes API from outside the cluster.

Via Ingress

By configuring the `apiserver.publishAPI.ingress` parameters, you can publish the API server on a dedicated domain (for more details, see the section on service domains in the documentation).

During configuration, you can specify:

- The list of network addresses and subnets from which connections are allowed;
- The Ingress controller on which the publication occurs;
- Whether to use a manually provided TLS certificate, one obtained via cert-manager, or an automatic self-signed certificate.

By default, a special CA certificate will be generated and a kubeconfig generator will be automatically configured.

Via Gateway API

If the `alb` module is enabled in the cluster and a Gateway can be resolved for it (either a default Gateway discovered automatically, or one set explicitly via the `gatewayAPI` settings), the API server is additionally published via the Gateway API: a `ListenerSet` and an `HTTPRoute` are automatically created in the `kube-system` namespace, using the same hostname as for Ingress.

This publication method:

- Is controlled by the same `apiserver.publishAPI.ingress.enabled` parameter. There is no separate parameter to enable it.
- Requires the `global.modules.publicDomainTemplate` parameter to be set (for more details, see the section on service domains in the API documentation).
- Does not depend on whether the Ingress controller is enabled — it works as a fully independent mechanism alongside it.

> ⚠ If the Ingress controller and the `alb` module (with a resolvable Gateway) are enabled in the cluster at the same time, the API server gets published both via Ingress and via the Gateway API, under the same hostname. Only one of the two will actually be reachable by that name, depending on where its DNS record points. The other stays configured but unused. Don't disable the Ingress controller until you've confirmed that publication via the Gateway API works as expected.

Via a Service of type LoadBalancer

By configuring the `apiserver.publishAPI.loadBalancer` parameters, you can create a service of type LoadBalancer named `kube-system/d8-control-plane-apiserver`.

During configuration, you can specify:

- The list of network addresses and subnets from which connections are allowed;
- The external port of the service;
- Annotations on the service for load balancer provider settings.

Exposing the Kubernetes API over Ingress

The kube-apiserver component (without advanced configuration) is only accessible in the internal cluster network. This module enables easy and secure access to Kubernetes API from outside the cluster. The API server is exposed on a dedicated domain (for more details, see the section on service domains in the documentation).

When configuring, you can:

- list network addresses from which connection is allowed;
- list groups that are allowed to access the API server;
- specify Ingress-controller to authenticate on.

By default, a special CA certificate will be generated and the kubeconfig generator will be automatically configured.

Auditing

Kubernetes Auditing can help you if you need to keep track of operations in your Namespaces or troubleshoot the cluster. You can configure it by setting the appropriate Audit Policy. As the result you will have the log file `/var/log/kube-audit/audit.log` containing audit events according to the configured Policy.

By default, in a cluster with DP, a basic policy is created for logging events:

- related to the creation, deletion, and changing of resources;
- committed from the names of ServiceAccounts from the "system" Namespace `kube-system`, `d8-*`;
- committed with resources in the "system" Namespace `kube-system`, `d8-*`.

A basic policy can be disabled by setting the basicAuditPolicyEnabled flag to `false`.

When OIDC authentication is configured, additional user information is included in audit logs under the `user.extra` field:

- `user-authn.deckhouse.io/name` — user's display name
- `user-authn.deckhouse.io/preferred username` — user's preferred username
- `user-authn.deckhouse.io/dex-provider` — Dex provider identifier (requires `federated:id` scope)
- `user-authn.deckhouse.io/sub` — `sub` claim of the token: the user and the Dex provider in one identifier
- `user-authn.deckhouse.io/generation` — generation of the user's Dex account the token was issued at; the `user-authn` module raises it when it revokes the user's tokens

You can find how to set up policies in a special FAQ section.

Control plane monitoring

The module organizes secure metrics collection and provides a basic set of monitoring rules for the following cluster components:

- `kube-apiserver`;
- `kube-controller-manager`;
- `kube-scheduler`;
- `kube-etcd`.

Admission plugins enabled by default

When installing the Deckhouse Platform, in addition to the standard admission plugins enabled by Kubernetes, the module enables several additional ones. For more information about admission plugins, see the Kubernetes documentation.

Standard admission plugins enabled by Kubernetes

Kubernetes enables the following standard admission plugins:

Admission plugin	Type	Short description
NamespaceLifecycle	Validating	Prevents the creation of new objects in namespaces that are in the process of being deleted (termination) or in non-existent namespaces. It also prevents the deletion of system namespaces: `default`, `kube-system`, and `kube-public`.
LimitRanger	Mutating and Validating	Monitors incoming requests and checks whether they violate the limits specified in the LimitRange object within the namespace. It is used to enforce resource limits at the container and pod levels.
ServiceAccount	Mutating and Validating	Automates the handling of ServiceAccounts. If a pod does not specify a `ServiceAccount`, it automatically assigns the `default` account from the same namespace. It also verifies that the specified ServiceAccount exists.
TaintNodesByCondition	Mutating	A standard security plugin that automatically adds taints to newly created nodes based on their status (e.g., `NotReady` or `Unreachable`). This prevents pods from being scheduled on new nodes before their labels are updated to accurately reflect their declared state.
PodSecurity	Validating	Checks new pods before they are launched and determines whether to allow them based on the requested security context and the restrictions set by the Pod Security Standards in the namespace where the pod will reside.
Priority	Mutating and Validating	Uses the `priorityClassName` field and populates the integer priority value. If the priority class is not found, the pod is rejected.
DefaultTolerationSeconds	Mutating	Sets default toleration values for Pods for the `notready:NoExecute` and `unreachable:NoExecute` taints if the Pod does not have its own tolerations. The default value is **5 minutes**.
DefaultStorageClass	Mutating	Oversees the creation of PersistentVolumeClaim objects. If a specific StorageClass is not specified in the request, it automatically adds the StorageClass labeled `default`. This ensures that users who do not specify a StorageClass will receive the default StorageClass.
StorageObjectInUseProtection	Mutating	Protects storage objects (such as PersistentVolumes) that are in use by pods from accidental deletion. Adds the `kubernetes.io/pvc-protection` or `kubernetes.io/pv-protection` finalizers, which prevent the resource from being deleted while it is in use.

Admission plugin	Type	Short description
PersistentVolumeClaimResize	Validating	Performs additional checks on incoming requests to resize a PersistentVolumeClaim. By default, it prohibits resizing of all claims, except in cases where the claim's StorageClass explicitly allows resizing by setting the `allowVolumeExpansion` parameter to `true`.
RuntimeClass	Mutating and Validating	Takes the `RuntimeClass` into account when creating pods. Sets the `pod.Spec.Overhead` field according to the selected runtime class and validates requests.
CertificateApproval	Validating	Monitors requests to approve `CertificateSigningRequests` and performs additional authorization checks to ensure the user has the rights to approve certificate requests.
CertificateSigning	Validating	Monitors updates to the `status.certificate` field in a CertificateSigningRequest and verifies that the user has the rights to sign the certificate request, using the `spec.signerName` value specified in the CertificateSigningRequest resource.
ClusterTrustBundleAttest	Validating	Analyzes and attests to the trustworthiness of the Kubernetes cluster. This may include verifying certificates, security configurations, and other parameters related to the cluster's integrity.
CertificateSubjectRestriction	Validating	Monitors the creation of CertificateSigningRequests where `spec.signerName = kubernetes.io/kube-apiserver-client`. Rejects requests that specify the `system:masters` group.
DefaultIngressClass	Mutating	Monitors the creation of Ingress objects. If no Ingress class is specified in the request, it automatically adds the Ingress class labeled as "default."
PodTopologyLabels	Mutating	Adds topology labels (e.g., availability zone) to pods bound to a node, corresponding to the labels of that node.
MutatingAdmissionPolicy	Validating and Mutating	A mechanism within the admission control system that allows objects to be modified when they are created or updated during the request admission process.
MutatingAdmissionWebhook	Mutating	Invokes all mutating webhooks that correspond to the request. Webhooks can modify the object and are invoked sequentially.
ValidatingAdmissionPolicy	Validating	Allows declarative validation to be embedded directly into the API, without using external HTTP calls. It checks the CEL for incoming requests that meet the criteria. It is enabled when both the `validatingadmissionpolicy` feature and the `admissionregistration.k8s.io/v1alpha1` group/version are active. If any of the `ValidatingAdmissionPolicy` policies fail, the request is rejected.
ValidatingAdmissionWebhook	Validating	Invokes all validating webhooks that match the request to check the object.
ResourceQuota	Validating	Monitors incoming requests and checks whether they violate the limits specified in the ResourceQuota object in the namespace. Used to limit the total consumption of resources (CPU, memory, number of objects) in the namespace.

Additional admission plugins enabled by the module

In addition to the standard admission plugins enabled by Kubernetes, the module enables the following admission plugins (which cannot be disabled):

Admission plugin	Type	Short description
EventRateLimit	Validating	Addresses the issue of API server overload caused by requests to save new events. Allows you to configure limits at the namespace, user, or global level.
ExtendedResourceToleration	Mutating	Automatically adds tolerations to pods requesting extended resources (e.g., GPU, FPGA). This allows you to allocate special nodes for such pods—nodes that are pre-tainted with the resource name—without manually adding tolerations to the pods.
NodeRestriction	Validating	Restricts the set of Node and Pod objects that the kubelet can modify. Enhances cluster security.
PodNodeSelector	Mutating and Validating	Defines and restricts which node selectors can be used within a namespace, based on reading the namespace annotation and global configuration.
PodTolerationRestriction	Mutating and Validating	Checks the pod's toleration for conflicts with tolerations specified at the namespace level. If there are no conflicts, it merges the pod's and namespace's tolerations. It also checks the pod against a "whitelist" of tolerations.

i In addition to the admission plugins listed above (which are enabled by Kubernetes and the module), you can also enable some additional ones. To do this, use the `apiserver.admissionPlugins` parameter.

The `PodNodeSelector` and `PodTolerationRestriction` plugins are still listed among the allowed values of that parameter for backward compatibility only: they are always enabled, so specifying them has no effect and causes no error.

Feature Gates

You can configure feature gates using the enabledFeatureGates parameter of the `control-plane-manager` ModuleConfig.

Changing the list of feature gates causes a restart of the corresponding component (for example, `kube-apiserver`, `kube-scheduler`, `kube-controller-manager`, `kubelet`).

The following example enables the `ComponentFlagz` and `ComponentStatusz` feature gates:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    enabledFeatureGates:
      - ComponentFlagz
      - ComponentStatusz
```

If a feature gate is not supported or is deprecated, the monitoring system generates the D8ProblematicFeatureGateInUse alert indicating that the feature gate will not be applied.

⚠ The Kubernetes version update (controlled by the kubernetesVersion parameter) will not occur if the list of enabled feature gates for the new version of Kubernetes includes deprecated feature gates.

More information about feature gates is available in the Kubernetes documentation.

▾ List of available feature gates...

Kubernetes 1.33

- `AllowParsingUserUIDFromCertAuth`
- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ContainerStopSignals`

- `CrossNamespaceVolumeDataSource`
- `DeploymentReplicaSetTerminatingReplicas`
- `DisableAllocatorDualWrite`
- `HPAConfigurableTolerance`
- `HPAScaleToZero`
- `ImageVolume`
- `KubeletPSI`
- `KubeletPodResourcesGet`
- `ListFromCacheSnapshot`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `PodLogsQuerySplitStreams`
- `PodObservedGenerationTracking`
- `PreferSameTrafficDistribution`
- `ResourceHealthStatus`
- `SELinuxMount`
- `StrictIPCIDRValidation`

Kubernetes 1.34

- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ContainerRestartRules`
- `ContainerStopSignals`
- `CrossNamespaceVolumeDataSource`
- `DeploymentReplicaSetTerminatingReplicas`
- `EnvFiles`
- `HPAConfigurableTolerance`
- `HPAScaleToZero`
- `HostnameOverride`
- `ImageVolume`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `PodLogsQuerySplitStreams`
- `ResourceHealthStatus`
- `SELinuxMount`
- `StrictIPCIDRValidation`

Kubernetes 1.35

- `AllowUnsafeMalformedObjectDeletion`
- `CRDObservedGenerationTracking`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ConstrainedImpersonation`
- `ContainerStopSignals`
- `CrossNamespaceVolumeDataSource`
- `HPAScaleToZero`
- `MemoryQoS`
- `MutablePVNodeAffinity`
- `MutablePodResourcesForSuspendedJobs`
- `MutableSchedulingDirectivesForSuspendedJobs`
- `MutatingAdmissionPolicy`
- `NodeDeclaredFeatures`
- `PodLogsQuerySplitStreams`
- `ResourceHealthStatus`
- `RestartAllContainersOnContainerExits`
- `SELinuxMount`
- `StrictIPCIDRValidation`
- `TaintTolerationComparisonOperators`

Kubernetes 1.36

- `AllowUnsafeMalformedObjectDeletion`
- `CRDObservedGenerationTracking`
- `CRIListStreaming`
- `ConcurrentWatchObjectDecode`
- `ContainerStopSignals`
- `ControllerManagerReleaseLeaderElectionLockOnExit`
- `CrossNamespaceVolumeDataSource`
- `HPAScaleToZero`
- `MemoryQoS`
- `MutablePVNodeAffinity`
- `NativeHistograms`
- `PersistentVolumeClaimUnusedSinceTime`
- `PodLogsQuerySplitStreams`
- `SELinuxMount`
- `ShardedListAndWatch`
- `TaintTolerationComparisonOperators`

Kubernetes 1.37

- `AtomicWriteVolumeUserFields`
- `CRDObservedGenerationTracking`
- `CRIListStreaming`
- `ContainerStopSignals`
- `ControllerManagerReleaseLeaderElectionLockOnExit`
- `CrossNamespaceVolumeDataSource`
- `EmptyDirVolumeMode`
- `GRPCContainerProbeTLS`
- `H2CContainerProbe`
- `InPlacePodVerticalScalingMemoryBackedVolumes`
- `MutablePVNodeAffinity`
- `PodLogsQuerySplitStreams`
- `ShardedListAndWatch`
- `StatefulSetRecreateStrategy`
- `TaintTolerationComparisonOperators`

Protecting sensitive fields in custom resources

The `CRDSensitiveData` feature gate provides field-level protection for sensitive data in resources marked with the `x-kubernetes-sensitive-data: true` annotation. This feature is implemented as a patch to `kube-apiserver` (`apiextensions-apiserver`) and is supported starting from Kubernetes version 1.31.

The `x-kubernetes-sensitive-data` marker is validated by `kube-apiserver` when applying a resource:

- Marker requires the `CRDSensitiveData` feature gate to be enabled. It is enabled by default and shouldn't be specified manually.
- Marker can't be set on the root of the schema (the `openAPIV3Schema` node). To protect all fields of a resource, add the marker to the `spec` property (or a subtree below it), not to the schema root — the root also includes system fields (`apiVersion`, `kind`, `metadata`), which cannot be encrypted.
- Field type must be one of the OpenAPI v3 types: `string`, `integer`, `number`, `boolean`, `object`, or `array`. Applying the marker to `object` or `array` makes the entire subtree sensitive.
- Fields defined with `x-kubernetes-int-or-string: true` are supported.
- Marker is not allowed inside `anyOf`, `oneOf`, `allOf`, or `not` branches (this is enforced by the structural schema validator).

If at least one field in the resource schema is marked with `x-kubernetes-sensitive-data: true`, the following protection mechanisms are applied to all custom resources of this type:

- **Encryption in etcd**: Entire resource is encrypted using the same mechanism as Kubernetes Secrets. Requires enabling the `apiserver.encryptionEnabled` parameter.
- **RBAC-based field filtering**: For `get`, `list`, and `watch` requests, sensitive fields are removed from API responses if the caller does not have the corresponding permissions on the `<resource>/sensitive` subresource.
- **Audit log masking**: Values of sensitive fields are replaced with `"******"` in audit logs, regardless of RBAC permissions and audit level.

To add encryption in etcd to sensitive data protection, set the `apiserver.encryptionEnabled` parameter to `true`. The `CRDSensitiveData` feature gate is enabled by default and it shouldn't be specified manually:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true
```

⚠ Enabling `encryptionEnabled` is irreversible and triggers a `kube-apiserver` restart.

For details, see the following sections:

- FAQ: Instructions for enabling sensitive data protection.
- Examples: Configuration examples and results.

Third-party components

List of third-party software used in the `control-plane-manager` module:

- Kubernetes 1.33+

  License: Apache License 2.0

  An open source system for managing containerized applications across multiple hosts.

- Etcd 3.6.15

  License: Apache License 2.0

  A distributed reliable key-value store for the most critical data of a distributed system.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

Some cluster parameters that affect control plane management historically came from the ClusterConfiguration resource. Prefer the corresponding settings in this ModuleConfig instead — for example, `kubernetesVersion`. The ClusterConfiguration fields remain only as a deprecated fallback during migration.

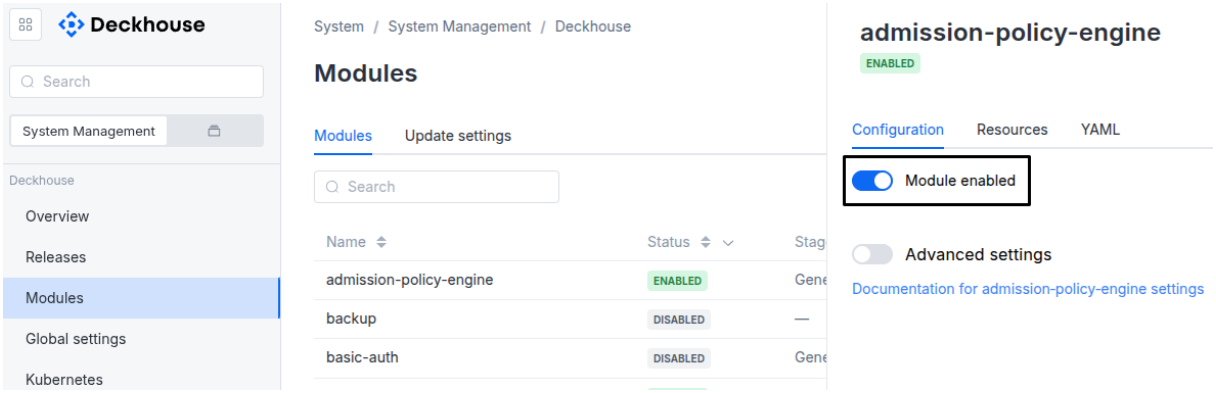
The module has 36 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `control-plane-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable control-plane-manager
```

- **Using ModuleConfig `control-plane-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `control-plane-manager` (create it if necessary);

  Example of a manifest to enable module `control-plane-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
```

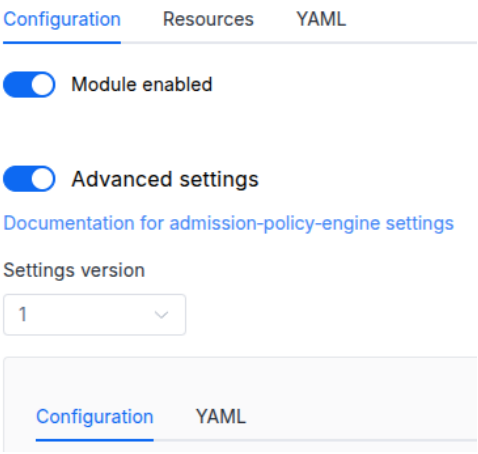
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `control-plane-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `control-plane-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `control-plane-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `control-plane-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc control-plane-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `control-plane-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `control-plane-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: control-plane-manager
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `etcd.externalMembersNames`

- Updates from version **2** to **3**:

  Replace `.apiserver.loadBalancer` with `.apiserver.publishAPI.loadBalancer`.

Parameters

Schema version: 3

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    resourcesRequests:
      cpu: 1000m
      memory: 500Mi
    apiserver:
      bindToWildcard: true
      certSANs:
      - bakery.infra
      - devs.infra
      publishAPI: {}
```

- **settings**
  object
  - **settings.apiserver**
    object

    `kube-apiserver` parameters.

    - **settings.apiserver.admissionPlugins**
      array of strings

      List of enabled additional admission plugins.

      Note, that in addition to the admission plugins enabled by default in Kubernetes, the following admission plugins are also always enabled:

      - `ExtendedResourceToleration`
      - `PodNodeSelector`
      - `PodTolerationRestriction`
      - `NodeRestriction`
      - `EventRateLimit` with the following config:

        ```
        apiVersion: eventratelimit.admission.k8s.io/v1alpha1
        kind: Configuration
        limits:
        - type: Namespace
          qps: 50
          burst: 100
          cacheSize: 2000
        ```

      > **i** Note that the `PodNodeSelector` admission plugin does not require specifying a global configuration, it relies on annotated Namespaces.

      During DP installation, Kubernetes enables the standard admission plugins. In addition to these, the module enables several more. For details, see the “Admission plugins enabled by default” section.

      The `PodNodeSelector` and `PodTolerationRestriction` values are kept in the list of allowed values for backward compatibility only: these plugins are always enabled, so specifying them has no effect and causes no error.

      Example:

      ```
      admissionPlugins:
      - AlwaysPullImages
      - NamespaceAutoProvision
      ```

      - **Element of the array**
        string

        Allowed values: `AlwaysPullImages`, `NamespaceAutoProvision`, `OwnerReferencesPermissionEnforcement`, `PodNodeSelector`, `PodTolerationRestriction`

    - **settings.apiserver.auditLog**
      object

      Audit policy settings

      Default: `{"output":"File"}`

      - **settings.apiserver.auditLog.output**
        string

        Required value

        Audit logs target stream.

Default: `File`

Allowed values: `File`, `Stdout`

Example:

```
output: Stdout
```

- **settings.apiserver.auditLog.path**
  string

  Directory path for logs if the output is “File”, otherwise ignored.

  Default: `/var/log/kube-audit`

  Pattern: `^[a-zA-Z0-9_/.-]+[a-zA-Z0-9_.-]$`

- **settings.apiserver.auditPolicyEnabled**
  boolean

  Set the audit policies using the configuration from the `kube-system/audit-policy` Secret.

  Default: `false`

- **settings.apiserver.authn**
  object

  Optional authentication parameters for Kubernetes API clients.

  By default, they are taken from user-authn module ConfigMap.

  - **settings.apiserver.authn.oidcCA**
    string

    OIDC provider CA.

  - **settings.apiserver.authn.oidcIssuerAddress**
    string

    OIDC provider network address alias.

    Examples:

    ```
    oidcIssuerAddress: 1.2.3.4
    ```

    ```
    oidcIssuerAddress: ''
    ```

  - **settings.apiserver.authn.oidcIssuerURL**
    string

    OIDC provider URL.

    Example:

    ```
    oidcIssuerURL: https://my-super-site.tech/
    ```

  - **settings.apiserver.authn.webhookCA**
    string

    Authorization webhook CA.

  - **settings.apiserver.authn.webhookCacheTTL**
    string

    The duration to cache responses from the webhook token authenticator.

    It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

    Example:

    ```
    webhookCacheTTL: 5m
    ```

  - **settings.apiserver.authn.webhookURL**
    string

    Authentication webhook URL.

    Example:

    ```
    webhookURL: https://127.0.0.1:40443/
    ```

- **settings.apiserver.authz**
  object

  Optional authorization parameters for Kubernetes API clients.

  By default, they are taken from user-authz module ConfigMap.

  - **settings.apiserver.authz.webhookCA**

string

Authorization webhook CA.

- **settings.apiserver.authz.webhookURL**
  string

  Authorization webhook URL.

  Example:

  ```
  webhookURL: https://127.0.0.1:40443/
  ```

- **settings.apiserver.basicAuditPolicyEnabled**
  boolean

  Enforce basic Deckhouse audit policies.

  Default: `true`

- **settings.apiserver.bindToWildcard**
  boolean

  Specifies whether to listen on `0.0.0.0`.

  By default, the API server listens on the hostIP. The latter usually corresponds to the Internal node address; however, the actual IP depends on the cluster type (Static or Cloud) and the layout selected.

  Default: `false`

- **settings.apiserver.certSANs**
  array of strings

  Array of SANs, with which the API server certificate will be generated.

  In addition to the passed list, the following list is always used:

  - `kubernetes`;
  - `kubernetes.default`;
  - `kubernetes.default.svc`;
  - `kubernetes.default.svc.cluster.local`;
  - Kubernetes API server address;
  - `127.0.0.1`;
  - host name;
  - host IP.

  Example:

  ```
  certSANs:
  - my-site.com
  - 192.168.67.76
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9a-zA-Z\.-]+$`

- **settings.apiserver.encryptionEnabled**
  boolean

  Enables data encryption in etcd (see Encrypting secret data at rest).

  When enabled, a Secret `kube-system/d8-secret-encryption-key` is created to store the encryption key.

  Sensitive custom resource fields marked with `x-kubernetes-sensitive-data: true` are protected by the `CRDSensitiveData` feature gate for `kube-apiserver`, which is enabled by default. Enabling this parameter adds encryption in etcd to RBAC-based field filtering with the `<resource>/sensitive` subresource and hiding values from the audit log.

  > **Warning.** Once enabled, this parameter can’t be disabled.

  Default: `false`

- **settings.apiserver.publishAPI**
  object

  Publishing access to Kubernetes API.

  - **settings.apiserver.publishAPI.ingress**
    object

    Settings for PublishAPI type Ingress.

    - **settings.apiserver.publishAPI.ingress.addKubeconfigGeneratorEntry**
      boolean

      Setting it to `false` will remove an entry in kubeconfig-generator.

      Default: `true`

    - **settings.apiserver.publishAPI.ingress.enabled**

boolean

Setting it to `true` will create an Ingress resource `kubernetes-api` in the `kube-system` namespace in the cluster (it exposes the Kubernetes API).

Default: `false`

- **settings.apiserver.publishAPI.ingress.https**
  object

  The HTTPS mode for the API server Ingress.

  Examples:

  ```
  https:
    mode: SelfSigned
  ```

  ```
  https:
    mode: Global
    global:
      kubeconfigGeneratorMasterCA: plainstring
  ```

  - **settings.apiserver.publishAPI.ingress.https.global**
    object

    An additional parameter for the `Global` mode.

    - **settings.apiserver.publishAPI.ingress.https.global.kubeconfigGeneratorMasterCA**
      string

      If there is an external load balancer in front of the Ingress that terminates HTTPS traffic using non-public CA, then you need to specify the CA so it will be included in kubectl-config.

      If you are using certificates issued by the `cert-manager` module and Let’s Encrypt in your cluster, you should set an empty string `""` as the value.

      Also, you can set the external LB’s certificate itself as a CA if you can’t get the CA that signed it for some reason. Note that after the certificate is updated on the LB, all the previously generated kubeconfigs will stop working.

  - **settings.apiserver.publishAPI.ingress.https.mode**
    string

    The mode of issuing certificates for the Ingress resource.

    In the `SelfSigned` mode, a CA-signed certificate will be issued for the Ingress resource.

    Use the following command to get the certificate: `d8 k -n kube-system get secrets kubernetes-api-ca-key-pair -oyaml`.

    In the `Global` mode, the policies specified in the `global.modules.https.mode` global parameter will be applied. Thus, if the global parameter has the `CertManager` mode set (with `letsencrypt` as the ClusterIssuer), then the Let’s Encrypt certificate will be issued for the Ingress resource.

    Default: `SelfSigned`

    Allowed values: `SelfSigned`, `Global`

  - **settings.apiserver.publishAPI.ingress.ingressClass**
    string

    The Ingress class that will be used to expose the Kubernetes API via Ingress.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **settings.apiserver.publishAPI.ingress.whitelistSourceRanges**
    array of strings

    An array of CIDRs that are allowed to connect to the API server.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))?$`

- **settings.apiserver.publishAPI.loadBalancer**
  object

  Settings type LoadBalancer

  - **settings.apiserver.publishAPI.loadBalancer.annotations**
    object

    Annotations to attach to a service to fine-tune the load balancer.

> **Caution!** The module does not take into account the specifics of setting annotations in various cloud environments. If the annotations for load balancer provisioning are only applied when creating a service, you will need to delete and add the `apiserver.loadBalancer` parameter to update such parameters.

- **settings.apiserver.publishAPI.loadBalancer.enabled**
  boolean

  If set to `true`, service `kube-system/d8-control-plane-apiserver` of the LoadBalancer type will be created.

  Default: `false`

- **settings.apiserver.publishAPI.loadBalancer.port**
  integer

  External LoadBalancer TCP port.

  Default: `443`

  Allowed values: `1 <= X <= 65534`

- **settings.apiserver.publishAPI.loadBalancer.sourceRanges**
  array of strings

  A list of CIDRs that are allowed to connect to the API.

  The cloud provider may not support this option or ignore it.

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\/[0-9]+$`

- **settings.apiserver.serviceAccount**
  object

  ServiceAccount issuing settings.

  Examples:

  ```
  serviceAccount: {}
  ```

  ```
  serviceAccount:
    additionalAPIAudiences:
    - istio-ca
  ```

  - **settings.apiserver.serviceAccount.additionalAPIAudiences**
    array of strings

    A list of API audiences to add when provisioning ServiceAccount tokens.

    The default audiences list includes:

    - `serviceAccount.issuer`, if specified.
    - `serviceAccount.additionalAPIIssuers`, if specified.
    - `https://kubernetes.default.svc.${clusterDomain}`, for example, `https://kubernetes.default.svc.cluster.local` – this audience is always added for backward compatibility.

    The service account token authenticator will validate that tokens used against the API are bound to at least one of these audiences.

    This option is beneficial when migrating from one API issuer to another.

  - **settings.apiserver.serviceAccount.additionalAPIIssuers**
    array of strings

    A list of additional issuers to include when provisioning ServiceAccount tokens.

    Issuers (`iss`) are used to verify the source of the tokens, ensuring they originate from trusted entities.

    The first issuer is automatically generated based on the template `https://kubernetes.default.svc.${clusterDomain}`, for example, `https://kubernetes.default.svc.cluster.local`. When multiple issuers are specified, the first issuer is used to generate tokens, and all provided issuers are accepted for token verification.

    This option is beneficial when migrating from one API issuer to another.

  - **settings.apiserver.serviceAccount.issuer**
    string

    ServiceAccount issuer. This is the URL of the API server. The values of this field are used as the `iss` claim of the token and to verify Service Account JWT tokens.

    Note, all pods in the cluster using ServiceAccount tokens must be restarted upon changing this option.

    Example:

    ```
    issuer: https://api.example.com
    ```

- **settings.enabledFeatureGates**
  array of strings

  List of feature gates enabled in the control plane.

  More information about feature gates is available in the Kubernetes documentation. The list of feature gates supported in DP can be found on the module overview page.

  Default: `[]`

  - **Element of the array**
    string

    Pattern: `^[a-zA-Z][a-zA-Z0-9]*$`

- **settings.encryptionAlgorithm**
  string

  Asymmetric encryption algorithm used when generating keys and certificates for the following control-plane components:

  - `apiserver`;
  - `apiserver-kubelet-client`;
  - `apiserver-etcd-client`;
  - `front-proxy-client`;
  - `etcd-server`;
  - `etcd-peer`;
  - `etcd-healthcheck-client`.

  Certificates for the components listed above will be reissued using the selected algorithm and key length.

  If parameter not set, the value from `ClusterConfiguration.encryptionAlgorithm` is used (defaults to `RSA-2048`).

  > **Warning.** When reissuing certificates, the root certificate (**CA**) is not rotated. The root certificate is created with the selected algorithm only during the initial cluster bootstrap.

  Default: `RSA-2048`

  Allowed values: `RSA-2048`, `RSA-3072`, `RSA-4096`, `ECDSA-P256`

  Example:

  ```
  encryptionAlgorithm: ECDSA-P256
  ```

- **settings.etcd**
  object

  `etcd` parameters.

  - **settings.etcd.backup**
    object

    etcd backup parameters.

    - **settings.etcd.backup.cronSchedule**
      string

      Backup schedule etcd in cron format. The local time zone of the `kube-controller-manager` is used.

      Backups on different master nodes are started with a 15-minute offset from each other to avoid simultaneous disk load on all etcd members. Nodes are processed in reverse order of their names, so the node with the lowest name (for example, `master-0`) is backed up last and stores the most recent snapshot.

      Default: `0 0 * * *`

      Example:

      ```
      cronSchedule: 0 1 * * *
      ```

    - **settings.etcd.backup.enabled**
      boolean

      Enable etcd backup.

      Default: `true`

    - **settings.etcd.backup.hostPath**
      string

      Host path on masters nodes for etcd backup.

      Default: `/var/lib/etcd`

  - **settings.etcd.defrag**
    object

    etcd defragmentation parameters.

    Default: `{"cronSchedule":"0 1 * * *"}`

    - **settings.etcd.defrag.cronSchedule**

string

Defragmentation schedule in cron format (UTC).

Default: `0 1 * * *`

Pattern: `^(*(\/[0-9]+)?|[0-9]+(-[0-9]+)?(\/[0-9]+)?(,[0-9]+(-[0-9]+)?(\/[0-9]+)?)*)\s+(*(\/[0-9]+)?|[0-9]+(-[0-9]+)?(\/[0-9]+)?(,[0-9]+(-[0-9]+)?(\/[0-9]+)?)*)\s+(*(\/[0-9]+)?|[0-9]+(-[0-9]+)?(\/[0-9]+)?(,[0-9]+(-[0-9]+)?(\/[0-9]+)?)*)\s+(*(\/[0-9]+)?|[0-9]+(-[0-9]+)?(\/[0-9]+)?(,[0-9]+(-[0-9]+)?(\/[0-9]+)?)*)$`

Example:

```
cronSchedule: 0 1 * * *
```

- **settings.etcd.defrag.enabled**
  boolean

  Enable periodic etcd defragmentation.

  Defaults to `true` when the cluster has three or more etcd members (masters ≥ 3, or masters = 2 with an etcd-arbiter node); otherwise defaults to `false`. An explicit value always takes priority over the computed default.

  > **Warning.** Enabling defragmentation on a single-master cluster will cause a brief control plane unavailability while etcd is being defragmented.

- **settings.etcd.maxDbSize**
  number

  quota-backend-bytes parameter. Deckhouse automatically manages the `quota-backend-bytes` parameter. If the `maxDbSize` parameter is set, deckhouse will use this value for the `quota-backend-bytes` etcd parameter.

  Minimum: 512MB.

  Maximum: 8GB.

  Experimental. It can be removed in the future.

  Allowed values: `536870912 <= X <= 8589934592`

- **settings.failedNodePodEvictionTimeoutSeconds**
  integer

  The number of seconds after which pods will be deleted from the node with the `Unreachable` status.

  > **Note!** If you change the parameter, the pods must be restarted.

  Default: `300`

- **settings.kubernetesVersion**
  string

  Desired Kubernetes version of the control plane components.

  This is the preferred place to manage the cluster Kubernetes version. Edit with:

  ```
  d8 k edit mc control-plane-manager
  ```

  The cluster Kubernetes version is resolved in the following order: this parameter, then the deprecated `ClusterConfiguration.kubernetesVersion`, then the version the current Deckhouse release defaults to. This parameter wins whenever it is set, `Default` included; while it is unset, the deprecated field still decides.

  If `Default` is specified, DP selects the version considered stable for the current release. If that version is more than 1 minor version below the maximum version that has ever been used in the cluster, the version is not changed automatically.

  Downgrading more than 1 minor version below the highest version the cluster has ever run is forbidden.

  Allowed values: `1.33`, `1.34`, `1.35`, `1.36`, `1.37`, `Default`

  Examples:

  ```
  kubernetesVersion: '1.35'
  ```

  ```
  kubernetesVersion: Default
  ```

- **settings.network**
  object

  Cluster network parameters.

  Each parameter is resolved in the following order: this section, then the deprecated same-named field in `ClusterConfiguration`. A parameter set here wins; while it is unset, the deprecated field still decides.

> **Warning.** `podSubnetCIDR`, `serviceSubnetCIDR` and `podSubnetNodeCIDRPrefix` describe the network of a running cluster. Changing any of them after the cluster is bootstrapped is **extremely dangerous** and can lead to complete cluster failure, so it is blocked by admission. It is **strongly recommended to recreate the cluster** instead. See documentation for details. `clusterDomain` is not subject to this warning: changing it is a normal, documented procedure.

- **settings.network.clusterDomain**
  string

  Cluster domain (used for local routing).

  Set the cluster domain here. The same field in `ClusterConfiguration` is deprecated and will be removed in a future release.

  Please note: the domain must not match the domain used in the DNS name template in the publicDomainTemplate parameter. For example, you cannot set `clusterDomain: cluster.local` and `publicDomainTemplate: %s.cluster.local` at the same time.

  When moving the value from `ClusterConfiguration`, keep it exactly as it is now: a different value is rejected. To change the domain later, follow the documented procedure.

  Default: `cluster.local`

  Pattern: `^[0-9a-zA-Z._-]+$`

  Example:

  ```
  clusterDomain: cluster.local
  ```

- **settings.network.podSubnetCIDR**
  string

  Address space of the cluster's Pods.

  When migrating from the deprecated `ClusterConfiguration.podSubnetCIDR`, the value must match the one the cluster currently runs with; admission rejects a mismatch.

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

  Example:

  ```
  podSubnetCIDR: 10.111.0.0/16
  ```

- **settings.network.podSubnetNodeCIDRPrefix**
  string

  The prefix size of the Pod network allocated to each node.

  This parameter determines how many IP addresses are available for Pods on each individual node. For example, if `podSubnetCIDR: 10.111.0.0/16` and `podSubnetNodeCIDRPrefix: "24"`, each node will receive a `/24` subnet (e.g., `10.111.0.0/24`, `10.111.1.0/24`, etc.), providing up to 254 IP addresses per node (253 usable for Pods, accounting for network and broadcast addresses).

  Calculation examples:

  - With `/16` network and `/24` prefix: maximum **256 nodes** (2^(24-16) = 256)
  - With `/16` network and `/25` prefix: maximum **512 nodes**, but only **126 IPs per node**
  - Each Pod network IP can support up to 2 Pods (one running + one terminating)

  If the parameter is set in neither this section nor the deprecated `ClusterConfiguration.podSubnetNodeCIDRPrefix`, `24` is used.

  When migrating from the deprecated field, the value must match the one the cluster currently runs with; admission rejects a mismatch.

  Pattern: `^[0-9]{1,2}$`

  Example:

  ```
  podSubnetNodeCIDRPrefix: '24'
  ```

- **settings.network.serviceSubnetCIDR**
  string

  Address space of the cluster's services.

  When migrating from the deprecated `ClusterConfiguration.serviceSubnetCIDR`, the value must match the one the cluster currently runs with; admission rejects a mismatch.

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

  Example:

  ```
  serviceSubnetCIDR: 10.222.0.0/16
  ```

- **settings.nodeMonitorGracePeriodSeconds**
  integer

  The number of seconds after which the node will enter the `Unreachable` status in case of lost connection.

Default: `40`

- **settings.resourcesRequests**
  object

  The amount of CPU and memory resources allocated to control-plane components on each master node.

  If CPU or memory requests are not explicitly specified, the module automatically calculates requests for the corresponding resource for each control plane component once a day based on the average usage over the previous seven days, plus a 10% margin so that actual usage stays below the request. Explicitly specifying the total amount of CPU or memory disables automatic request calculation for that resource for all control plane components.

  The explicitly specified amount is distributed among the components in the following fixed proportions:

  - `kube-apiserver` — 45%;
  - `etcd` — 35%;
  - `kube-controller-manager` — 10%;
  - `kube-scheduler` — 10%.

  For example, `cpu: 1000m` results in requests of `450m` for `kube-apiserver`, `350m` for `etcd`, and `100m` for each of `kube-controller-manager` and `kube-scheduler`.

  These settings do not apply if the cluster control plane is managed by a cloud provider (for example, Google Kubernetes Engine or Azure Kubernetes Service). More info in the Kubernetes documentation.

  Example:

  ```
  resourcesRequests:
    cpu: 1000m
    memory: 500Mi
  ```

  - **settings.resourcesRequests.cpu**
    string or number

    The combined CPU requests for control-plane components on each master node.

    Pattern: `^[0-9]+m?$`

  - **settings.resourcesRequests.memory**
    string

    The combined memory requests for control-plane components on each master node.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.rootKubeconfigSymlink**
  boolean

  Whether to maintain a symlink from `/root/.kube/config` to `/etc/kubernetes/admin.conf` on control-plane nodes.

  When the user-authz module is enabled and this parameter is `false`, the symlink is removed and the `NODE_ADMIN_KUBECONFIG` environment variable is set on control-plane-manager pods accordingly. Use `d8 k` with `--kubeconfig=/etc/kubernetes/admin.conf` (or other credentials) when you need API access from the node.

  When the `user-authz` module is disabled, `control-plane-manager` ignores this parameter and keeps the default (symlink present).

  Setting the value back to `true` restores the symlink on the next reconciliation.

  Default: `true`

  Examples:

  ```
  rootKubeconfigSymlink: true
  ```

  ```
  rootKubeconfigSymlink: false
  ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

ControlPlaneNode

Short names: cpn

Scope: Namespaced
Version: v1alpha1

Describes the desired and observed state of control plane components on a single node: etcd, kube-apiserver, kube-controller-manager, kube-scheduler.

The resource is created and updated by the `control-plane-manager` module automatically. Users do not need to create or edit it.

Useful for diagnosing the health of the control plane on the node:

- `ETCD`, `APISERVER`, `CONTROLLERMANAGER` and `SCHEDULER` columns of `kubectl get cpn` show component readiness.
- `CERTIFICATES` column shows overall certificate health.
- `spec.components` contains desired fingerprints, while `status.components` contains the actually applied ones.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Describes the desired state of control plane components on the node.

  - **spec.caChecksum**
    string

    Checksum of the `d8-pki` Secret containing CA certificates that must be applied to all components on the node.

  - **spec.components**
    object

    Contains the desired configuration and PKI checksums for each control plane component.

    If a value in `spec.components.<component>.checksums` differs from the corresponding value in `status.components.<component>.checksums`, the module starts a ControlPlaneOperation to bring the component to the desired state.

    - **spec.components.etcd**
      object

      Desired state of the etcd component.

      - **spec.components.etcd.checksums**
        object

        Desired checksums for the component.

        - **spec.components.etcd.checksums.ca**
          string

          Checksum of the CA certificates applied to the component.

          Absent in `spec` because CA configuration is global.

          Set in `status` after the component pod restarts with the new CA certificates.

        - **spec.components.etcd.checksums.config**
          string

          Checksum of the component static pod manifest and associated configuration files.

        - **spec.components.etcd.checksums.pki**
          string

          Checksum of the component PKI settings (`certSANs`, `encryption-algorithm`).

    - **spec.components.kube-apiserver**
      object

      Desired state of the kube-apiserver component.

      - **spec.components.kube-apiserver.checksums**
        object

        Desired checksums for the component.

        - **spec.components.kube-apiserver.checksums.ca**
          string

          Checksum of the CA certificates applied to the component.

          Absent in `spec` because CA configuration is global.

          Set in `status` after the component pod restarts with the new CA certificates.

        - **spec.components.kube-apiserver.checksums.config**
          string

          Checksum of the component static pod manifest and associated configuration files.

        - **spec.components.kube-apiserver.checksums.pki**
          string

Checksum of the component PKI settings (`certSANs` , `encryption-algorithm`).

- **spec.components.kube-controller-manager**
  object

  Desired state of the kube-controller-manager component.

  - **spec.components.kube-controller-manager.checksums**
    object

    Desired checksums for the component.

    - **spec.components.kube-controller-manager.checksums.ca**
      string

      Checksum of the CA certificates applied to the component.

      Absent in `spec` because CA configuration is global.

      Set in `status` after the component pod restarts with the new CA certificates.

    - **spec.components.kube-controller-manager.checksums.config**
      string

      Checksum of the component static pod manifest and associated configuration files.

    - **spec.components.kube-controller-manager.checksums.pki**
      string

      Checksum of the component PKI settings (`certSANs` , `encryption-algorithm`).

- **spec.components.kube-scheduler**
  object

  Desired state of the kube-scheduler component.

  - **spec.components.kube-scheduler.checksums**
    object

    Desired checksums for the component.

    - **spec.components.kube-scheduler.checksums.ca**
      string

      Checksum of the CA certificates applied to the component.

      Absent in `spec` because CA configuration is global.

      Set in `status` after the component pod restarts with the new CA certificates.

    - **spec.components.kube-scheduler.checksums.config**
      string

      Checksum of the component static pod manifest and associated configuration files.

    - **spec.components.kube-scheduler.checksums.pki**
      string

      Checksum of the component PKI settings (`certSANs` , `encryption-algorithm`).

- **status**
  object

  Describes the observed state of control plane components on the node.

  - **status.caChecksum**
    string

    Actually applied checksum of CA certificates.

  - **status.components**
    object

    Contains the observed state for each component: applied checksums and certificate expiration dates.

    - **status.components.etcd**
      object

      Observed state of the etcd component.

      - **status.components.etcd.certificatesExpirationTime**
        object

        Maps each component certificate file name to its `NotAfter` timestamp.

        Populated during the `CertObserve` step.

      - **status.components.etcd.checksums**
        object

        Applied checksums for the component.

        - **status.components.etcd.checksums.ca**
          string

          Checksum of the CA certificates applied to the component.

Absent in `spec` because CA configuration is global.

Set in `status` after the component pod restarts with the new CA certificates.

- **status.components.etcd.checksums.config**
  string

  Applied checksum of the component static pod manifest and associated configuration files.

- **status.components.etcd.checksums.pki**
  string

  Applied checksum of the component PKI settings (`certSANs`, `encryption-algorithm`).

- **status.components.etcd.lastCertObserveTime**
  string

  Time of the last successful `CertObserve` step for the component.

- **status.components.kube-apiserver**
  object

  Observed state of the kube-apiserver component.

  - **status.components.kube-apiserver.certificatesExpirationTime**
    object

    Maps each component certificate file name to its `NotAfter` timestamp.

    Populated during the `CertObserve` step.

  - **status.components.kube-apiserver.checksums**
    object

    Applied checksums for the component.

    - **status.components.kube-apiserver.checksums.ca**
      string

      Checksum of the CA certificates applied to the component.

      Absent in `spec` because CA configuration is global.

      Set in `status` after the component pod restarts with the new CA certificates.

    - **status.components.kube-apiserver.checksums.config**
      string

      Applied checksum of the component static pod manifest and associated configuration files.

    - **status.components.kube-apiserver.checksums.pki**
      string

      Applied checksum of the component PKI settings (`certSANs`, `encryption-algorithm`).

  - **status.components.kube-apiserver.lastCertObserveTime**
    string

    Time of the last successful `CertObserve` step for the component.

- **status.components.kube-controller-manager**
  object

  Observed state of the kube-controller-manager component.

  - **status.components.kube-controller-manager.certificatesExpirationTime**
    object

    Maps each component certificate file name to its `NotAfter` timestamp.

    Populated during the `CertObserve` step.

  - **status.components.kube-controller-manager.checksums**
    object

    Applied checksums for the component.

    - **status.components.kube-controller-manager.checksums.ca**
      string

      Checksum of the CA certificates applied to the component.

      Absent in `spec` because CA configuration is global.

      Set in `status` after the component pod restarts with the new CA certificates.

    - **status.components.kube-controller-manager.checksums.config**
      string

      Applied checksum of the component static pod manifest and associated configuration files.

    - **status.components.kube-controller-manager.checksums.pki**
      string

      Applied checksum of the component PKI settings (`certSANs`, `encryption-algorithm`).

  - **status.components.kube-controller-manager.lastCertObserveTime**

string

Time of the last successful `CertObserve` step for the component.

- **status.components.kube-scheduler**
  object

  Observed state of the kube-scheduler component.

  - **status.components.kube-scheduler.certificatesExpirationTime**
    object

    Maps each component certificate file name to its `NotAfter` timestamp.

    Populated during the `CertObserve` step.

  - **status.components.kube-scheduler.checksums**
    object

    Applied checksums for the component.

    - **status.components.kube-scheduler.checksums.ca**
      string

      Checksum of the CA certificates applied to the component.

      Absent in `spec` because CA configuration is global.

      Set in `status` after the component pod restarts with the new CA certificates.

    - **status.components.kube-scheduler.checksums.config**
      string

      Applied checksum of the component static pod manifest and associated configuration files.

    - **status.components.kube-scheduler.checksums.pki**
      string

      Applied checksum of the component PKI settings (`certSANs`, `encryption-algorithm`).

  - **status.components.kube-scheduler.lastCertObserveTime**
    string

    Time of the last successful `CertObserve` step for the component.

- **status.conditions**
  array of objects

  List of conditions reflecting the readiness of control plane components on the node.

  Possible condition types:

  - `EtcdPodReady`, `APIServerPodReady`, `ControllerManagerPodReady`, `SchedulerPodReady`: the component's static pod is running and ready.
  - `EtcdInSync`, `APIServerInSync`, `ControllerManagerInSync`, `SchedulerInSync`: the desired configuration is applied to the component (`True` with reason `InSync`). `False` with reason `OperationInQueue` means an operation is waiting for approval, `OperationInFlight` means an operation is running (the `message` field contains the operation name(s)), `OutOfSync` means there is no active operation, but the applied configuration differs from the desired one (for example, in maintenance mode).
  - `EtcdReady`, `APIServerReady`, `ControllerManagerReady`, `SchedulerReady`: `<Component>PodReady` and `<Component>InSync` are both `True` (`ETCD`, `APISERVER`, `CONTROLLERMANAGER`, `SCHEDULER` columns). When `False`, `reason` and `message` are taken from the failing condition; a not ready pod takes precedence.
  - `CertificatesHealthy`: All component certificates are valid and have enough lifetime left (`CERTIFICATES` column).

  The `<Component>PodReady` conditions reflect only the health of the component's static pod. Whether the desired configuration is applied to the component is reported by `<Component>InSync`.

  When `status` is `False`, the cause is described in `reason` and `message`.

  - **status.conditions.lastTransitionTime**
    string

    Required value

    Time of the last condition status transition.

  - **status.conditions.message**
    string

    Required value

    Human-readable description of the current condition state.

    Maximum length: `32768`

  - **status.conditions.observedGeneration**
    integer

    The `.metadata.generation` value used to calculate the current condition state.

    Allowed values: `0 <= X`

  - **status.conditions.reason**
    string

    Required value

Machine-readable reason for the current condition state.

Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Status of the condition.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Type of the condition in CamelCase or in the `foo.example.com/CamelCase` format.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

ControlPlaneOperation

Short names: cpo

Scope: Namespaced
Version: v1alpha1

ControlPlaneOperation describes a single update operation for a control plane component on a specific node — for example, certificate renewal or applying a new manifest.

The resource is created and updated by the control-plane-manager module automatically; users do not need to create or edit it.

Useful for diagnostics: shows which operation is running on the node, which step is currently active and whether the operation completed successfully (see Phase and CurrentStep columns of `kubectl get cpo`).

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Describes the desired state of an operation.

  - **spec.approved**
    boolean

    Required value

    Indicates whether the operation is allowed to run.

    Only one approved operation may run on a node at a time. The approver controller sets this automatically based on the current control plane state.

    Default: `false`

  - **spec.component**
    string

    Required value

    Control plane component the operation targets.

    Allowed values: `Etcd`, `KubeAPIServer`, `KubeControllerManager`, `KubeScheduler`

  - **spec.desiredCaChecksum**
    string

    Expected CA certificate checksum once the operation completes.

    Populated only for steps that update the root CA certificate (`SyncCA` and related).

  - **spec.desiredConfigChecksum**
    string

    Expected component configuration checksum (static pod manifest and associated configuration files) once the operation completes.

    Used to confirm that the operation actually applied the intended configuration. After the pod restarts the checksum in `ControlPlaneNode.status` must match this value.

  - **spec.desiredPkiChecksum**
    string

    Expected component PKI checksum (`certSANs`, `encryption-algorithm`) once the operation completes.

    Populated only for steps that change PKI (`RenewPKICerts` and related).

- **spec.nodeName**
  string

  Required value

  Name of the control plane node on which the operation must be executed.

- **spec.steps**
  array of strings

  Required value

  Ordered list of steps to perform within the operation.

  Possible values:

  - `Backup`: Backs up the current component configuration (static pod manifest, associated configuration files) into `/etc/kubernetes/deckhouse/backup` before any changes.
  - `SyncCA`: Synchronizes CA certificates (`ca.crt`, `ca.key`) with the current `d8-pki` Secret.
  - `RenewPKICerts`: Re-issues component certificates (`server`, `peer`, `client`).
  - `RenewKubeconfigs`: Re-issues kubeconfig files used by the component to authenticate against the API.
  - `SyncManifests`: Updates the static pod manifest and accompanying files (`/etc/kubernetes/manifests`, associated configuration files) and records changes in `/etc/kubernetes/deckhouse/diffs` and `status.changedFiles`.
  - `JoinEtcdCluster`: Joins a new member to the etcd cluster.
  - `DefragEtcd`: Defragments the etcd data store on the target node to reclaim disk space.
  - `WaitPodReady`: Waits for the component static pod to become `Ready` after a restart.
  - `CertObserve`: Collects the current certificate expiration dates for the component and publishes them to `status.observedState`.
  - `RenewSignature`: Re-issues the signature key for the kube-apiserver.
  - **Element of the array**
    string

    Name of a single step performed within an operation.

    Allowed values: `Backup`, `SyncCA`, `RenewPKICerts`, `RenewKubeconfigs`, `SyncManifests`, `JoinEtcdCluster`, `DefragEtcd`, `WaitPodReady`, `CertObserve`, `RenewSignature`

- **status**
  object

  Observed state of an operation.

  - **status.changedFiles**
    array of objects

    Files that the `SyncManifests` step changed on the node.

    Full diffs of all files are kept on the node in `/etc/kubernetes/deckhouse/diffs`.

    File changed by an operation on the node.

    - **status.changedFiles.diff**
      string

      Unified diff of the file. Not published for files that may contain sensitive data.

      A diff larger than 32 KiB is truncated at a line boundary and ends with a line that says so.

      Maximum length: `32768`

    - **status.changedFiles.path**
      string

      Required value

      Absolute path of the file on the node.

  - **status.conditions**
    array of objects

    Reflects the operation progress.

    The primary condition is `Completed`. Its `reason` field is shown in the `Phase` column of `kubectl get cpo`:

    - `InProgress`: Operation is running. The current step name is shown in the `CurrentStep` column.
    - `Succeeded`: Operation finished successfully.
    - `Failed`: Operation finished with an error. See details in `message`.
    - **status.conditions.lastTransitionTime**
      string

      Required value

      Last time the condition transitioned from one status to another.

    - **status.conditions.message**
      string

      Required value

      Human-readable description of the current condition state. If there is an error, this field contains the last error output on a step.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**

integer

The `.metadata.generation` value that the condition was set based upon.

Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  Machine-readable reason for the current condition state (for example, `InProgress`, `Succeeded`, or `Failed`).

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Status of the condition.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Type of condition.

  In addition to `Completed`, a separate condition is created for each executed step, where `type` equals the step name (for example, `RenewPKICerts`, `SyncManifests`).

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.observedState**
  object

  Contains the component state collected by the `CertObserve` step.

  Populated only for static pod components (etcd, kube-apiserver, kube-controller-manager, kube-scheduler).

  - **status.observedState.certificatesExpirationTime**
    object

    Maps each component certificate file name to its `NotAfter` timestamp.

    Used by the module for certificate re-issuing and alerting.

KubeSchedulerWebhookConfiguration

Scope: Cluster
Version: v1alpha1

Defines the configuration for connecting a third-party kube-scheduler extender webhook.

An external kube-scheduler webhook allows expanding the capabilities of the scheduler and consider more complex conditions when planning the load in the Kubernetes cluster.

kube-scheduler metrics and the explanations of its decisions (the `Scheduled` event text and the decision log, as `extender/<name>`) name the extender of a webhook by the name of this object; when the object has several webhooks, the name gets `/<n>`, the position of the webhook in `webhooks` starting from 1.

During a simulation, kube-scheduler evaluates which nodes could host a Pod without actually placing it. The `filter` and `prioritize` requests sent to the extender include `dryRun: true`, which is absent from regular scheduling requests. When `dryRun` is `true`, the extender must compute its response as usual without causing side effects: it must not reserve space or resources for the Pod or modify its own state. An extender that rejects unknown JSON fields (for example, Go `DisallowUnknownFields`) fails simulation requests; regular scheduling requests are unaffected.

- **webhooks**
  array of objects

  Required value

  - **webhooks.clientConfig**
    object

    Required value

    ClientConfig defines how to communicate with the webhook.

    - **webhooks.clientConfig.caBundle**
      string

      Required value

      caBundle is a PEM encoded CA bundle which will be used to validate the webhook’s server certificate.

      Pattern: `^(?:[A-Za-z0-9+\/]{4})*(?:[A-Za-z0-9+\/]{4}|[A-Za-z0-9+\/]{3}=|[A-Za-z0-9+\/]{2}={2})$`

- **webhooks.clientConfig.service**
  object

  Required value

  Access to webhook via cluster service.

  - **webhooks.clientConfig.service.name**
    string

    Required value

    Service name.

    Pattern: `^[0-9a-z\-]+$`

    Length: `1..255`
  - **webhooks.clientConfig.service.namespace**
    string

    Required value

    Service namespace.

    Pattern: `^[0-9a-z\-]+$`

    Length: `1..255`
  - **webhooks.clientConfig.service.path**
    string

    Required value

    URI path.

    Pattern: `^[0-9a-zA-Z._\/-]+$`

    Length: `1..255`
  - **webhooks.clientConfig.service.port**
    integer

    Required value

    Service port.

    Allowed values: `1 <= X <= 65535`

- **webhooks.failurePolicy**
  string

  Specifies scheduling should fail or not fail when the extender returns an error or is not reachable.

  Default: `Fail`

  Allowed values: `Fail`, `Ignore`
- **webhooks.filterVerb**
  string

  Verb appended to `URLPrefix` when issuing a `filter` call to the scheduler extender. If omitted, defaults to `filter` (for backward compatibility, avoid making changes unless necessary). Set to an empty string `""` to disable `filter` calls for this extender.
- **webhooks.preemptVerb**
  string

  Verb appended to `URLPrefix` when issuing a `preempt` call to this extender. Leave empty (`""`) to disable `preempt` calls for this extender.
- **webhooks.prioritizeVerb**
  string

  Verb appended to `URLPrefix` when issuing a `prioritize` call to this extender. If omitted, defaults to `prioritize` (for backward compatibility, avoid making changes unless necessary). Set to an empty string `""` to disable `prioritize` calls for this extender.
- **webhooks.sendPlacements**
  boolean

  Send the extender information about tentative Pod reservations on nodes and their cancellations through the `assumed` field in `prioritize` requests. Each entry has the fields `uid`, `namespace`, `name`, `node` and `forget: true` for a canceled reservation.

  Requires `prioritizeVerb`. Entries older than 30 seconds are not sent. After a failed request, entries may be sent again. The extender must handle repeated entries and `forget` notifications for Pods it does not know, and expire reservations after 30 seconds if no `forget` arrives.

  Default: `false`
- **webhooks.timeout**
  string

  Webhook timeout: an integer with the `ms` or `s` suffix, for example `500ms` or `30s`. Allowed values are from `1ms` to `60s`.

Takes precedence over `timeoutSeconds`. If omitted, the `timeoutSeconds` value is used.

Pattern: `^[0-9]+(ms|s)$`

Example:

```
timeout: 500ms
```

- **webhooks.timeoutSeconds**
  Deprecated
  integer

  Webhook timeout in seconds.

  Deprecated. Use the `timeout` parameter instead. Ignored if `timeout` is set.

  Default: `10`

  Allowed values: `1 <= X <= 60`

- **webhooks.weight**
  integer

  Required value

  The numeric multiplier for the node scores that the prioritize call generates.

  Allowed values: `1 <= X`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How do I add a master node to a static or hybrid cluster?

> It is important to have an odd number of masters to ensure a quorum.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

When installing Deckhouse Platform with default settings, the NodeGroup `master` lacks the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Because of this, after changing the number of `staticInstances` nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), when adding a regular node using Cluster API Provider Static (CAPS), it can be "intercepted" and added to the NodeGroup `master`, even if the corresponding `StaticInstance` (in `metadata`) specifies a label with a `role` different from `master`. To avoid this "interception", after installing DP, edit the NodeGroup `master` — add the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Example of NodeGroup `master` with `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Next, when adding master nodes to the cluster using CAPS, specify the label specified in `spec.staticInstances.labelSelector` NodeGroup `master` in the corresponding `StaticInstance`. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # The label specified in spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

i When adding new master nodes using CAPS and changing the number of master nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), please note the following:

During the initial cluster installation, the configuration specifies the first master node on which the installation takes place. If, after the initial cluster installation, you need to create a multi-master cluster and add master nodes using CAPS, you must specify the number of nodes in the `spec.staticInstances.count` parameter of the NodeGroup `master` as one less than the desired number.

For example, if you need to create a multi-master with three master nodes in `spec.staticInstances.count` NodeGroup `master`, specify the value `2` and create two `staticInstances` for the nodes to be added. After adding them to the cluster, the number of master nodes will be three: the master node on which the installation took place and two master nodes added using CAPS.

Otherwise, adding a master node to a static or hybrid cluster is similar to adding a regular node. To do this, use the corresponding examples. All the necessary actions to configure cluster control plane components on the new master nodes are performed automatically. Wait until the master nodes appear in `Ready` status.

How do I add master nodes to a cloud cluster?

The following describes the conversion of a single-master cluster into a multi-master.

i Before adding nodes, ensure you have the required quotas in the cloud provider.

It is important to have an odd number of masters to ensure a quorum.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the creation of new master nodes.
4. Make sure that Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command to check the state before working:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
   <MASTER-NODE-0-HOST>
   ```

The command output should indicate that Terraform found no inconsistencies and no changes are required.

8. **In the installer container**, run the following command and specify the required number of replicas using the `masterNodeGroup.replicas` parameter:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST>
```

> **i** For **Yandex Cloud**, when using external addresses on master nodes, the number of array elements in the masterNodeGroup.instanceClass.externalIPAddresses parameter must equal the number of master nodes. If `Auto` is used (public IP addresses are provisioned automatically), the number of array elements must still equal the number of master nodes.
>
> To illustrate, with three master nodes (`masterNodeGroup.replicas: 3`) and automatic address reservation, the `masterNodeGroup.instanceClass.externalIPAddresses` parameter would look as follows:
>
> ```
> externalIPAddresses:
> - "Auto"
> - "Auto"
> - "Auto"
> ```

9. **In the installer container**, run the following command to start scaling:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

10. Wait until the required number of master nodes are `Ready` and all `control-plane-manager` instances are up and running:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-control-plane-manager
```

How do I reduce the number of master nodes in a cloud cluster?

The following describes the conversion of a multi-master cluster into a single-master.

> ⚠ The steps described below must be performed from the first in order of the master node of the cluster (master-0). This is because the cluster is always scaled in order: for example, it is impossible to delete nodes master-0 and master-1, leaving master-2.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DP queue is empty:

```
d8 system queue list
```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

```
docker login registry.deckhouse.io
```

During the login process, you will need to enter your `Username` and `Password`.

> **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. On the **local machine**, run the DP installer container for the corresponding edition and version (change the container registry address if needed):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command and set `masterNodeGroup.replicas` to `1`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
  --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

> **i**
>
> For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

8. **In the installer container**, run the following command to trigger the scaling operation:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

> **i**
>
> For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in Openstack itself.
>
> For example, when deleting the `cloud-demo-master-2` node in the Openstack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
>
> If the kubernetes-data disk remains, there may be problems with ETCD operation as the number of master nodes increases.

9. Check the Deckhouse queue and make sure that there are no errors with the command:

```
d8 system queue list
```

How do I dismiss the master role while keeping the node?

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty:

```
d8 system queue list
```

5. Remove the following labels:
   - `node-role.kubernetes.io/control-plane`
   - `node-role.kubernetes.io/master`
   - `node.deckhouse.io/group`

   Use the following command to remove labels:

```
d8 k label node <MASTER-NODE-N-NAME> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master- node.deckhouse.io/group-
```

6. Make sure that the master node to be deleted is no longer listed as a member of the etcd cluster:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

7. Exec to the node and run the following commands:

```
rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
rm -f /etc/kubernetes/authorization-webhook-config.yaml
rm -f /etc/kubernetes/admin.conf /root/.kube/config
rm -rf /etc/kubernetes/deckhouse
rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
rm -rf /var/lib/etcd/member/
```

How do I switch to a different OS image in a multi-master cluster?

The OS change method depends on the cluster type: in a cloud cluster, nodes are replaced with `dhctl converge`; in a static cluster, nodes are replaced manually, one at a time.

In a cloud cluster

To change the OS image in a cloud cluster with multiple master nodes, follow these steps:

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command to check the state before working:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   The command output should indicate that Terraform found no inconsistencies and no changes are required.

8. **In the installer container**, run the following command and specify the required OS image using the `masterNodeGroup.instanceClass` parameter (specify the addresses of all master nodes using the `--ssh-host` parameter):

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
   user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

9. **In the container with the installer**, run the following command to update the nodes:

   Carefully review the actions that converge plans to perform when it asks for confirmation.

   When the command is executed, the nodes will be replaced by new nodes with confirmation on each node. The replacement will be performed one by one in reverse order (2,1,0).

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   The following steps should be performed **sequentially on each** master node, starting with the node with the highest number (suffix 2) and ending with the node with the lowest number (suffix 0).

10. **On the newly created node**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

    ```
    journalctl -fu bashible.service
    ```

11. Make sure the node is listed as an etcd cluster member:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

12. Make sure `control-plane-manager` is running on the node:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
  -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
```

13. Proceed to update the next node (repeat the steps above).

In a static cluster

The following instruction describes OS replacement on master nodes that were added to the cluster **manually** using the `bootstrap.sh` script. Perform the steps for each master node, one node at a time. Do not proceed to the next node until the current node has rejoined the cluster and is healthy.

⚠ If master nodes are managed by Cluster API Provider Static (CAPS) through StaticInstance resources, do not use this instruction. First delete the StaticInstance, install the required OS, and add the node to the `master` NodeGroup again.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

To change the OS of a manually added master node, follow these steps:

1. Make a backup of `etcd` and the `/etc/kubernetes` directory. If the `stronghold` module is enabled, make sure its data has also been backed up.
2. Check the cluster health and make sure there are no alerts or pending Deckhouse queue tasks:

```
d8 status
```

3. Remove the master node labels from the node:

```
d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
  node-role.kubernetes.io/master- node.deckhouse.io/group-
```

   where `<MASTER_NODE_NAME>`: Name of the master node being changed.

4. Make sure the node has been removed from the etcd cluster member list:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

5. Evict workloads from the node with `d8 k drain`:

```
d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
```

6. Force-delete any Pods remaining on the node:

```
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --force
```

7. Delete the Node object:

```
d8 k delete node <MASTER_NODE_NAME>
```

8. Clean up the DP data on the removed master node. This command irreversibly removes Kubernetes and DP data from the node. Before running it, make sure you have selected the correct node and created the required backups:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

9. Install the required OS on the node.
10. On one of the remaining master nodes in the cluster, retrieve and decode the script for adding a master node:

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
  -o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
```

11. Securely copy the `bootstrap.sh` file obtained in the previous step to the node being added and run it as `root` on that node:

    ```
    bash bootstrap.sh
    ```

12. **On the node being added**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

    ```
    journalctl -fu bashible.service
    ```

13. Wait until the node reaches the `Ready` status:

    ```
    d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
    ```

14. Make sure the node is listed as an etcd cluster member:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
        --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
        --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

15. Make sure `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
    ```

16. Make sure there are no alerts or pending queue tasks:

    ```
    d8 status
    ```

17. Repeat the procedure for the next master node.

How do I switch to a different OS image in a single-master cluster?

The method depends on the cluster type: first add extra master nodes, replace the OS in multi-master mode, then restore the original number of master nodes.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

In a cloud cluster

1. Convert your single-master cluster to a multi-master one, as described in the guide on adding master nodes to a cluster.
2. Change the OS on the master nodes according to the instructions.
3. Convert your multi-master cluster to a single-master one according to the guide on excluding master nodes from the cluster.

In a static cluster

1. Add extra master nodes according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Remove extra master nodes from the control plane role according to the instructions. Then delete them from the cluster with `d8 k delete node <MASTER_NODE_NAME>` and power off the corresponding servers.

How to configure HA mode with two master nodes and an arbiter node?

Deckhouse Platform allows you to configure HA mode with two master nodes and an arbiter node. This approach allows you to meet HA requirements in conditions of limited resources.

Only etcd is placed on the arbiter node, without the other control plane components. This node is used to ensure the etcd quorum.

Requirements for the arbiter node:

- At least 2 CPU cores
- At least 4 GB of RAM
- At least 8 GB of disk space for etcd

The network latency requirements for the arbiter node are similar to those for the master nodes.

Configuring in a cloud cluster

The example below applies to a cloud cluster with three master nodes. To configure HA mode with two master nodes and an arbiter node in a cloud cluster, you need to remove one master node from the cluster and add one arbiter node.

To do this, follow these steps:

⚠ The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2`.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (for example, to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DP queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Platform license key.

6. On the **local machine**, run the DP installer container for the corresponding edition and version (change the container registry address if needed):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
     --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   Change the cloud provider settings:

   - Set `masterNodeGroup.replicas` to `2`.
   - Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

     ```
     nodeGroups:
       - name: arbiter
         replicas: 1
         nodeTemplate:
           labels:
             node.deckhouse.io/etcd-arbiter: ""
           taints:
             - key: node.deckhouse.io/etcd-arbiter
               effect: NoSchedule
         zones:
           - europe-west3-b
         instanceClass:
           machineType: n1-standard-4
       # ... the rest of the manifest
     ```

   - Save your changes.

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look as follows:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

8. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST>
   ```

   > **i** **Important**. For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in OpenStack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the OpenStack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the `kubernetes-data` disk remains, there may be problems with etcd operation as the number of master nodes increases.

9. Check the Deckhouse queue with the following command and make sure there are no errors:

   ```
   d8 system queue list
   ```

Configuring in a static cluster

To configure HA mode with two master nodes and an arbiter node in a static cluster, follow these steps:

1. Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

   ```
   apiVersion: deckhouse.io/v1
     kind: NodeGroup
     metadata:
       name: arbiter
     spec:
       nodeType: Static
       nodeTemplate:
         labels:
           node.deckhouse.io/etcd-arbiter: ""
         taints:
           - key: node.deckhouse.io/etcd-arbiter
             effect: NoSchedule
     # ... the rest of the manifest
   ```

2. Add a node to the cluster that will be used as an arbiter node in a way that is convenient for you.
3. Ensure that the added arbiter node is in the list of etcd cluster members.
4. Remove one master node from the cluster.

How do I view the list of etcd members?

Option 1

Use the `etcdctl member list` command.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Warning. The last parameter in the output table shows etcd member is in `learner` state, is not in `leader` state.

Option 2

To obtain information about etcd cluster nodes in tabular form, use the `etcdctl endpoint status` command. For the leader, the `IS LEADER` column will show `true`.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

What if something went wrong?

During its operation, `control-plane-manager` automatically creates backups of configurations and data that may be useful in case of issues. These backups are stored in the `/etc/kubernetes/deckhouse/backup` directory. If errors or unforeseen situations occur during operation, you can use these backups to restore to the previous stable state.

What if the etcd cluster fails?

If the etcd cluster is not functioning and it cannot be restored from a backup, you can attempt to rebuild it from scratch by following the steps below.

1. First, on all nodes that are part of your etcd cluster, except for one, remove the `etcd.yaml` manifest located in the `/etc/kubernetes/manifests/` directory. This last node will serve as a starting point for the new multi-master cluster.
2. On the last node, edit etcd manifest `/etc/kubernetes/manifests/etcd.yaml` and add the parameter `--force-new-cluster` to `spec.containers.command`.
3. After the new cluster is ready, remove the `--force-new-cluster` parameter.

> ⚠ This operation is unsafe and breaks the guarantees given by the consensus protocol. Note that it brings the cluster to the state that was saved on the node. Any pending entries will be lost.

What if etcd restarts with an error?

This method may be necessary if the `--force-new-cluster` option doesn't restore etcd work. Such a scenario can occur during an unsuccessful converge of master nodes, where a new master node was created with an old etcd disk, changed its internal address, and other master nodes are absent. Symptoms indicating the need for this method include: the etcd container being stuck in an endless restart with the log showing the error: `panic: unexpected removal of unknown remote peer`.

1. Find the `etcdutl` utility on the master node and copy the executable to `/usr/local/bin/`:

   ```
   ETCD_PID=$(crictl inspect $(crictl ps --name etcd -q | head -1) | jq .info.pid)
   cp /proc/${ETCD_PID}/root/usr/bin/etcdutl /usr/local/bin/etcdutl
   chmod +x /usr/local/bin/etcdutl
   ```

2. Create a new etcd database snapshot from the current local snapshot (`/var/lib/etcd/member/snap/db`):

   ```
   etcdutl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
   --initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-urls=https://<ADDRESS>:2380 \
   --skip-hash-check=true --data-dir /var/lib/etcdtest
   ```

   where:

   - `<HOSTNAME>`: Name of the master node.
   - `<ADDRESS>`: Address of the master node.

3. Execute the following commands to use the new snapshot:

   ```
   cp -r /var/lib/etcd /tmp/etcd-backup
   rm -rf /var/lib/etcd
   mv /var/lib/etcdtest /var/lib/etcd
   ```

4. Locate the `etcd` and `api-server` containers:

   ```
   crictl ps -a | egrep "etcd|apiserver"
   ```

5. Remove the `etcd` and `api-server` containers:

   ```
   crictl rm <CONTAINER-ID>
   ```

6. Restart the master node.

What to do if the database volume of etcd reaches the limit set in quota-backend-bytes?

When the database volume of etcd reaches the limit set by the `quota-backend-bytes` parameter, it switches to "read-only"

mode. This means that the etcd database stops accepting new entries but remains available for reading data. You can tell that you are facing a similar situation by executing the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
  etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ \
  endpoint status -w table --cluster
```

If you see a message like `alarm:NOSPACE` in the `ERRORS` field, you need to take the following steps:

1. On **each master node** (from the list of the etcd cluster members obtained earlier), find the line with `--quota-backend-bytes` in the etcd pod manifest at `/etc/kubernetes/manifests/etcd.yaml` and double the value. If there is no such line, add one, for example: `- --quota-backend-bytes=8589934592`. This sets the limit to 8 GB.
2. Disarm the active alarm that occurred due to reaching the limit. To do this, execute the command:

   ```
   d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p)
   -- \
   etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
   --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
   --endpoints https://127.0.0.1:2379/ alarm disarm
   ```
3. Change the maxDbSize parameter in the `control-plane-manager` settings to match the value specified in the manifest.

How to compact etcd storage

> ⚠ Before compacting etcd storage, back up etcd.

To view the size of the etcd database on a specific node before and after compaction, use the command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  endpoint status --cluster -w table
```

Output example (the size of the etcd database on the node is specified in the `DB SIZE` column):

```
+----------------------------+------------------+---------+-----------------+---------+--------+-------
---------------+--------+------------+------------+-----------+------------+--------------------+------
--+----------------------------+-------------------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE | IN USE |
PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX |
ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+----------------------------+------------------+---------+-----------------+---------+--------+-------
---------------+--------+------------+------------+-----------+------------+--------------------+------
--+----------------------------+-------------------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |       true |      false |        56 |  258054684 |          258054684 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+-------
---------------+--------+------------+------------+-----------+------------+--------------------+------
--+----------------------------+-------------------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+-------
---------------+--------+------------+------------+-----------+------------+--------------------+------
--+----------------------------+-------------------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+-------
---------------+--------+------------+------------+-----------+------------+--------------------+------
--+----------------------------+-------------------+
```

How to compact etcd storage on a node in a single-master cluster

> ⚠ Compacting etcd storage is a resource-intensive operation that temporarily blocks etcd from running on that node. Keep this in mind when choosing a time to perform the operation in a cluster with a single master node.

To compact etcd storage in a cluster with a single master node, use the following command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `—command-timeout` parameter from the command above until compaction completes successfully.

How to compact etcd storage in a cluster with multiple master nodes

To compact etcd storage in a cluster with multiple master nodes:

1. Get a list of etcd pods. To do this, use the following command:

   ```
   d8 k -n kube-system get pod -l component=etcd -o wide
   ```

   Example output:

   ```
   NAME           READY    STATUS    RESTARTS   AGE     IP              NODE        NOMINATED NODE
   READINESS GATES
   etcd-master-0   1/1     Running   0          3d21h   192.168.199.80  master-0    <none>
   <none>
   etcd-master-1   1/1     Running   0          3d21h   192.168.199.81  master-1    <none>
   <none>
   etcd-master-2   1/1     Running   0          3d21h   192.168.199.82  master-2    <none>
   <none>
   ```

2. Identify the leader master node. To do this, contact any etcd pod and get a list of nodes participating in the etcd cluster using the command (where `NODE_NAME` is the name of the master node):

   ```
   d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
     --cert=/etc/kubernetes/pki/etcd/server.crt \
     --key=/etc/kubernetes/pki/etcd/server.key \
     --cacert=/etc/kubernetes/pki/etcd/ca.crt \
     endpoint status --cluster -w table
   ```

   Output example (the leader in the `IS LEADER` column will have the value `true`):

   ```
   +----------------------------+------------------+---------+-----------------+---------+--------+--
   --------------------+--------+------------+------------+-----------+------------+-----------------
   ---+--------+--------------------------+-------------------+
   |          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE | IN USE |
   PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED
   INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
   +----------------------------+------------------+---------+-----------------+---------+--------+--
   --------------------+--------+------------+------------+-----------+------------+-----------------
   ---+--------+--------------------------+-------------------+
   | https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
   20% | 2.1 GB |       true |      false |        56 |  258054684 |          258054684 |        |
   |             false |
   +----------------------------+------------------+---------+-----------------+---------+--------+--
   --------------------+--------+------------+------------+-----------+------------+-----------------
   ---+--------+--------------------------+-------------------+
   | https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
   20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
   |             false |
   +----------------------------+------------------+---------+-----------------+---------+--------+--
   --------------------+--------+------------+------------+-----------+------------+-----------------
   ---+--------+--------------------------+-------------------+
   | https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
   20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
   |             false |
   +----------------------------+------------------+---------+-----------------+---------+--------+--
   --------------------+--------+------------+------------+-----------+------------+-----------------
   ---+--------+--------------------------+-------------------+
   ```

3. Compact etcd storage on the etcd cluster member nodes one by one. Use the following command (where `NODE_NAME` is the name of the master node):

   > **i** Important: compact the leader node last.
   >
   > Restoring etcd on a node after compaction may take some time. It is recommended to wait at least a minute before proceeding to compact etcd storage on the next node.

   ```
   d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
     --cacert /etc/kubernetes/pki/etcd/ca.crt \
     --cert /etc/kubernetes/pki/etcd/ca.crt \
     --key /etc/kubernetes/pki/etcd/ca.key \
     --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
   ```

   Example output when the operation is successful:

   ```
   Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
   ```

   > **i** If a timeout error occurs, increase the value of the `—command-timeout` parameter from the command above until compaction completes successfully.

Cluster admin access model

The control-plane-manager module maintains several kubeconfig files on master nodes. Understanding their purpose is important for secure cluster administration.

Kubeconfig files on master nodes

File	Identity	Purpose
`/etc/kubernetes/admin.conf`	`kubernetes-admin` (`kubeadm:cluster-admins` group)	Machine kubeconfig for control-plane-manager operations (kubeconfig renewal, cluster administration). The group is bound to the built-in wildcard `cluster-admin` role.
`/etc/kubernetes/super-admin.conf`	`kubernetes-super-admin` (`system:masters` group)	Break-glass emergency credential. Bypasses RBAC entirely. Restrict access to this file to trusted recovery scenarios.
`/etc/kubernetes/controller-manager.conf`	`system:kube-controller-manager`	Used by kube-controller-manager.
`/etc/kubernetes/scheduler.conf`	`system:kube-scheduler`	Used by kube-scheduler.

RBAC-based admin access

`admin.conf` is generated with the `kubeadm:cluster-admins` group instead of `system:masters`. This provides RBAC-controlled admin access that can be revoked by removing the RBAC binding objects for `kubeadm:cluster-admins`.

Deckhouse binds the `kubeadm:cluster-admins` group to the built-in wildcard ClusterRole `cluster-admin` (the same effective model as a standard Kubernetes cluster without extra RBAC), regardless of whether the user-authz module is enabled. For full unrestricted access that also bypasses RBAC, use `super-admin.conf`.

Recommended admin access

The following options are available for administrative access:

- Personalized OIDC-based kubeconfig obtained through the DP web interface. This provides individual accountability and audit trail.
- Administrators can explicitly use the admin kubeconfig on a master node:

  ```
  d8 k --kubeconfig=/etc/kubernetes/admin.conf <command>
  ```

How to generate a kubeconfig to access the Kubernetes API?

`kubeconfig` for remote cluster access via `d8` (`kubectl`) can be generated in the DP web interface.

To publish the Kubernetes API and generate `kubeconfig`, configure the `apiserver.publishAPI` parameter of the `control-plane-manager` module:

- Open the `control-plane-manager` module settings (create the ModuleConfig `control-plane-manager` resource if there is none):

  ```
  d8 k edit mc control-plane-manager
  ```

- Add the following section to the `settings` block and save:

```
apiserver:
  publishAPI:
    ingress:
      enabled: true
```

Root kubeconfig symlink

By default, the CPM module creates a symlink `/root/.kube/config` -> `/etc/kubernetes/admin.conf` on master nodes, allowing root to run `d8 k` without specifying `--kubeconfig`.

When the **user-authz** module is enabled, you can disable this symlink by setting `rootKubeconfigSymlink: false` in the **control-plane-manager** module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

If **user-authz** is disabled, CPM does not apply this setting and keeps the default symlink behavior.

When the symlink is disabled (and **user-authz** is enabled), the symlink is removed when it pointed to `admin.conf`. Use personalized credentials or pass `--kubeconfig` explicitly.

Security hardening

The CPM module automatically restricts file permissions on `admin.conf` and `super-admin.conf` to `0600` (owner read/write only) during every reconciliation cycle. This prevents unauthorized users from reading these sensitive credentials.

Break-glass access

In emergency situations (RBAC misconfiguration, webhook failures), use `super-admin.conf`:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <command>
```

This credential bypasses all RBAC checks. Use it only as a last resort and restrict who can read the file.

How do I configure additional audit policies?

1. Enable the auditPolicyEnabled flag in the module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    apiserver:
      auditPolicyEnabled: true
```

2. Create the `kube-system/audit-policy` Secret containing a Base64 encoded YAML file:

```
apiVersion: v1
kind: Secret
metadata:
  name: audit-policy
  namespace: kube-system
data:
  audit-policy.yaml: <base64>
```

   The minimum viable example of the `audit-policy.yaml` file looks as follows:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  omitStages:
  - RequestReceived
```

You can find detailed information on how to configure `audit-policy.yaml` file here:

- The official Kubernetes documentation;
- The code of the generator script used in GCE.

How to omit Deckhouse built-in policy rules?

Set the apiserver.basicAuditPolicyEnabled module parameter to `false`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

How to stream audit log to stdout instead of files?

Set the apiserver.auditLog.output parameter to `Stdout`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

How to deal with the audit log?

There must be some `log scraper` on master nodes *(log-shipper, promtail, filebeat)* that will monitor the log file:

```
/var/log/kube-audit/audit.log
```

The following fixed parameters of log rotation are in use:

- The maximum disk space is limited to `1000 Mb`.
- Logs older than `30 days` will be deleted.

Depending on the `Policy` settings and the number of requests to the `apiserver`, the amount of logs collected may be high. Thus, in some cases, logs can only be kept for less than 30 minutes.

> ⚠ The current implementation of this feature isn't safe and may lead to a temporary failure of the control plane.
>
> The `apiserver` will not be able to start if there are unsupported options or a typo in the Secret.

If `apiserver` is unable to start, you have to manually disable the `--audit-log-*` parameters in the `/etc/kubernetes/manifests/kube-apiserver.yaml` manifest and restart apiserver using the following command:

```
docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')
# Or (depending on your CRI).
crictl stopp $(crictl pods --name=kube-apiserver -q)
```

After the restart, you will be able to fix the Secret or delete it:

```
d8 k -n kube-system delete secret audit-policy
```

How do I speed up the restart of Pods if the connection to the node has been lost?

By default, a node is marked as unavailable if it does not report its state for 40 seconds. After another 5 minutes, its Pods will be rescheduled to other nodes. Thus, the overall application unavailability lasts approximately 6 minutes.

In specific cases, if an application cannot run in multiple instances, there is a way to lower its unavailability time:

1. Reduce the period required for the node to become `Unreachable` if the connection to it is lost by setting the

`nodeMonitorGracePeriodSeconds` parameter.
2. Set a lower timeout for evicting Pods on a failed node using the `failedNodePodEvictionTimeoutSeconds` parameter.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 3
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```

In this case, if the connection to the node is lost, the applications will be restarted in about 1 minute.

Both these parameters directly impact the CPU and memory resources consumed by the control plane. By lowering timeouts, we force system components to send statuses more frequently and check the resource state more often.

When deciding on the appropriate threshold values, consider resources consumed by the control nodes (graphs can help you with this). Note that the lower parameters are, the more resources you may need to allocate to these nodes.

etcd backup and restore

What is done automatically

CronJob `kube-system/d8-etcd-backup-*` is automatically started at 00:00 UTC+0. Master nodes are backed up one by one with a 15-minute offset in reverse order of node names, so the node with the lowest name (for example, `master-0`) is backed up last and stores the most recent snapshot. The result is saved in `/var/lib/etcd/etcd-backup.tar.gz` on all nodes with `control-plane` in the cluster (master nodes). `/var/lib/etcd` is the default directory, it can be changed in the etcd.backup.hostPath parameter.

The archives on different nodes can contain snapshots taken at different times. If a backup job fails on a node, its archive can be even older. To restore, use the snapshot with the highest revision. To find out the revision, unpack the archive and run:

```
tar -xzf etcd-backup.tar.gz
etcdutl snapshot status etcd-backup.snapshot -w table
```

How to manually backup etcd

Using Deckhouse CLI (Deckhouse Platform v1.65+)

Starting with Deckhouse Platform v1.65, a new `d8 backup etcd` tool is available for taking snapshots of etcd state.

```
d8 backup etcd ./etcd-backup.snapshot
```

Using bash (Deckhouse Platform v1.64 and older)

Login into any control-plane node with `root` user and use next script:

```
#!/usr/bin/env bash
set -e

pod=etcd-`hostname`
d8 k -n kube-system exec "$pod" -- /usr/bin/etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert
/etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ snapshot save
/var/lib/etcd/${pod##*/}.snapshot && \
mv /var/lib/etcd/"${pod##*/}.snapshot" etcd-backup.snapshot && \
cp -r /etc/kubernetes/ ./ && \
tar -cvzf kube-backup.tar.gz ./etcd-backup.snapshot ./kubernetes/
rm -r ./kubernetes ./etcd-backup.snapshot
```

In the current directory etcd snapshot file `kube-backup.tar.gz` will be created from one of an etcd cluster members. From this file, you can restore the previous etcd cluster state in the future.

Also, we recommend making a backup of the `/etc/kubernetes` directory, which contains:

- manifests and configurations of control-plane components;
- Kubernetes cluster PKI.

This directory will help to quickly restore a cluster in case of complete loss of control-plane nodes without creating a new cluster and without rejoin the remaining nodes into the new cluster.

We recommend encrypting etcd snapshot backups as well as backup of the directory `/etc/kubernetes/` and saving them outside the Deckhouse cluster. You can use one of third-party files backup tools, for example: Restic, Borg, Duplicity, etc.

You can see documentation for learn about etcd disaster recovery procedures from snapshots.

How do perform a full recovery of the cluster state from an etcd backup?

The following steps will be described to restore to the previous state of the cluster from a backup in case of complete data loss.

Restoring a single-master cluster

Follow these steps to restore a single-master cluster on master node:

1. Find `etcdutl` utility on the master-node and copy the executable to `/usr/local/bin/`:

   ```
   ETCD_PID=$(crictl inspect $(crictl ps --name etcd -q | head -1) | jq .info.pid)
   cp /proc/${ETCD_PID}/root/usr/bin/etcdutl /usr/local/bin/etcdutl
   chmod +x /usr/local/bin/etcdutl
   ```

   Check the version of `etcdutl` using the command:

   ```
   etcdutl version
   ```

   The result must be a correct output of `etcdutl version` command without errors.

   Alternatively, you can download etcdutl executable to the node (preferably its version is the same as the etcd version in the cluster):

   ```
   wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
   tar -xzvf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdutl /usr/local/bin/etcdutl
   ```

   You can check current `etcd` version using following command (might not work, if etcd and Kubernetes API are already unavailable):

   ```
   d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdutl version
   ```

2. Stop the etcd.

   ```
   mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
   ```

3. Save the current etcd data.

   ```
   cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
   ```

4. Copy or move the `etcd-backup.snapshot` file to the current user’s (root) home directory:

   ```
   cp /path/to/backup/etcd-backup.snapshot ~/etcd-backup.snapshot
   ```

5. Clean the etcd directory.

   ```
   rm -rf /var/lib/etcd
   ```

6. Restore the etcd database.

   ```
   ETCDCTL_API=3 etcdutl snapshot restore ~/etcd-backup.snapshot  --data-dir=/var/lib/etcd
   ```

7. Run etcd. The process may take some time.

   ```
   mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
   crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
   ```

   To verify that etcd is running, use the command:

   ```
   crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
   ```

   Output example:

   ```
   CONTAINER        IMAGE            CREATED              STATE     NAME      ATTEMPT     POD ID
   POD
   4b11d6ea0338f    16d0a07aa1e26    About a minute ago   Running   etcd      0
   ee3c8c7d7bba6   etcd-gs-test
   ```

8. Restart the master node.

Restoring a multi-master cluster

To properly restore a multi-master cluster, follow these steps:

1. Enable High Availability (HA) mode. This is necessary to preserve at least one Prometheus replica and its PVC, since HA is disabled by default in single-master clusters.
2. Switch the cluster to single-master mode:
   - In a cloud cluster, follow the instructions.
   - In a static cluster, remove any unnecessary master nodes from the `control-plane` role by following the instructions and then remove them from the cluster.
   - In a static cluster with the configured HA mode based on two master nodes and an arbiter node, remove the arbiter node and additional master nodes.
   - In a cloud cluster with the configured HA mode based on two master nodes and an arbiter node, use the instructions to remove the additional master nodes and the arbiter node.
3. Restore etcd from the backup on the only remaining master node. Follow the instructions for restoring a cluster with a single master node.
4. Once etcd is restored, remove the records of the previously deleted master nodes from the cluster using the following command (replace with the actual node name):

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

   > **Warning.** If the `d8 k` or `kubectl` commands are unavailable on the node, check the `/etc/kubernetes/kubernetes-api-proxy/nginx.conf` configuration file. It should only specify your current API server. If the configuration contains lines with IP addresses of old master nodes, remove them. Edit the configuration in a similar way on all other nodes.

5. Reboot the master node. Ensure that the other nodes transition to the `Ready` state.
6. Wait for Deckhouse to process all tasks in the queue:

   ```
   d8 system queue main
   ```

7. Switch the cluster back to multi-master mode. Follow the respective instructions for cloud clusters and static clusters.

Once you go through these steps, the cluster will be successfully restored in the multi-master configuration.

How do I restore a Kubernetes object from an etcd backup?

To get cluster objects data from an etcd backup, you need:

1. Start an temporary instance of etcd.
2. Fill it with data from the backup.
3. Get desired objects using `auger`.

Example of steps to restore objects from an etcd backup

In the example below, `etcd-backup.snapshot` is an etcd snapshot, `infra-production` is the namespace in which objects need to be restored.

- To decode objects from `etcd` you would need auger. It can be built from source on any machine that has Docker installed (it cannot be done on cluster nodes).

  ```
  git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
  cd auger
  make release
  build/auger -h
  ```

- Resulting executable `build/auger`, and also the `snapshot` from the backup copy of etcd must be uploaded on master-node, on which following actions would be performed.

Following actions are performed on a master node, to which `etcd snapshot` file and `auger` tool were copied:

1. Set the correct access permissions for the backup file:

   ```
   chmod 644 etcd-backup.snapshot
   ```

2. Set full path for snapshot file and for the tool into environmental variables:

   ```
   SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot
   AUGER_BIN=/root/auger
   chmod +x $AUGER_BIN
   ```

3. Run a Pod with temporary instance of `etcd`.
   - Create Pod manifest. It should schedule on current master node by `$HOSTNAME` variable, and mounts snapshot file by `$SNAPSHOT` variable, which it then restores in temporary `etcd` instance:

```
cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
  - operator: Exists
  initContainers:
  - command:
    - etcdutl
    - snapshot
    - restore
    - "/tmp/etcd-snapshot"
    - --data-dir=/default.etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-snapshot-restore
    # Uncomment the fragment below to set limits for the container if the node does not have
enough resources to run it.
    # resources:
    #   requests:
    #     ephemeral-storage: "200Mi"
    #   limits:
    #     ephemeral-storage: "500Mi"
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
    - name: etcd-snapshot
      mountPath: /tmp/etcd-snapshot
      readOnly: true
  containers:
  - command:
    - etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-temp
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
  volumes:
  - name: etcddir
    emptyDir: {}
    # Use the snippet below instead of emptyDir: {} to set limits for the container if the
node's resources are insufficient to run it.
    # emptyDir:
    #  sizeLimit: 500Mi
  - name: etcd-snapshot
    hostPath:
      path: $SNAPSHOT
      type: File
EOF
```

- Create Pod from the resulting manifest:

```
d8 k create -f etcd.pod.yaml
```

4. Set environment variables. In this example:
   - `infra-production` - namespace which we will search resources in.
   - `/root/etcd-restore/output` - path for outputting recovered resource manifests.
   - `/root/auger` - path to `auger` executable.

```
FILTER=infra-production
BACKUP_OUTPUT_DIR=/root/etcd-restore/output
mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
```

5. Commands below will filter needed resources by `$FILTER` and output them into `$BACKUP_OUTPUT_DIR` directory:

```
files=($(d8 k -n default exec etcdrestore -c etcd-temp -- etcdctl  --endpoints=localhost:2379 get / --prefix --
keys-only | grep "$FILTER"))
for file in "${files[@]}"
do
  OBJECT=$(d8 k -n default exec etcdrestore -c etcd-temp -- etcdctl  --endpoints=localhost:2379 get "$file" --
print-value-only | $AUGER_BIN decode)
  FILENAME=$(echo $file | sed -e "s#/registry/##g;s#/#_#g")
  echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
  echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
done
```

6. From resulting `yaml` files, delete `creationTimestamp`, `UID`, `status` and other operational fields, and then restore the objects:

   ```
   d8 k create -f deployments_infra-production_supercronic.yaml
   ```

7. Delete the Pod with a temporary instance of etcd:

   ```
   d8 k -n default delete pod etcdrestore
   ```

How the node to run the Pod on is selected

The Kubernetes scheduler component selects the node to run the Pod on.

The selection process involves two phases, namely `Filtering` and `Scoring`. They are supposed to efficiently distribute the Pods between the nodes.

Although there are some additional phases, such as `pre-filtering`, `post-filtering`, and so on, you can safely narrow them down to the global phases mentioned above, as they merely increase flexibility and help to optimize things.

The structure of the Kubernetes scheduler

The Scheduler comprises plugins that function in either or both phases.

Example of plugins:

- **ImageLocality** — favors nodes that already have the container images that the Pod runs. Phase: `Scoring`.
- **TaintToleration** — implements taints and tolerations. Phases: `Filtering`, `Scoring`.
- **NodePorts** - checks whether the ports required for the Pod to run are available on the node. Phase: `Filtering`.

The full list of plugins is available in the Kubernetes documentation.

Working logic

Scheduler profiles

There are two predefined scheduler profiles:

- `default-scheduler`: The default profile that distributes pods to nodes with the lowest load;
- `high-node-utilization`: A profile that places pods on nodes with the highest load.

To specify a scheduler profile, use the `spec.schedulerName` parameter in the pod manifest.

Example of using a profile:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
  - name: example-pod
    image: registry.k8s.io/pause:2.0
```

Pod scheduling stages

The selection process starts with the `Filtering` phase. During it, `filter` plugins select nodes that satisfy filter conditions such as `taints`, `nodePorts`, `nodeName`, `unschedulable`, etc. If the nodes are in different zones, the scheduler alternates zones when selecting to ensure that all Pods will not end up in the same zone.

Suppose there are two zones with the following nodes:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

In this case, the nodes will be selected in the following order:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4.
```

Note that Kubernetes limits the number of nodes to calculate their scores during scheduling. This optimizes the selection process and prevents unnecessary scoring. By default, the threshold is linear. For clusters with less than or equal to 50 nodes, 100% of nodes are considered for scheduling; for clusters with 100 nodes, a 50%-threshold is used; and for clusters with 5000 nodes, a 10%-threshold is used. The minimum threshold value is 5% for clusters with more than 5000 nodes. Therefore, even if all the conditions are met, a node may not be included in the list of candidates for scheduling if the default settings are used.

This logic can be changed (read more about the parameter `percentage Of Nodes To Score` in the Kubernetes documentation), but Deckhouse does not provide such an option.

The `Scoring` phase follows once the nodes that meet the conditions are selected. Each plugin evaluates the filtered node list and assigns a score to each node based on available resources: `pod capacity`, `affinity`, `volume provisioning`, and other factors. The scores from the different plugins are then summed up and the node with the highest score is selected. If several nodes have the same score, the node is selected at random.

Finally, the scheduler assigns the Pod to the node with the highest ranking.

Documentation

- General description of the scheduler.
- Plugin system.
- Node Filtering Details.
- Scheduler source code.

How to change or extend the scheduler logic

To change the logic of the scheduler it is possible to use the extension mechanism Extenders.

Each plugin is a webhook that must satisfy the following requirements:

- Use of TLS.
- Accessibility through a service within the cluster.
- Support for standard `Verbs` (`filterVerb = filter`, `prioritizeVerb = prioritize`).
- It is also assumed that all plugins can cache node information (`nodeCacheCapable: true`).

You can connect an `extender` using KubeSchedulerWebhookConfiguration resource.

✖ When using the `failurePolicy: Fail` option, in case of an error in the webhook's operation, the scheduler will stop working and new pods will not be able to start.

kube-scheduler pauses an extender with `failurePolicy: Ignore` if 10 `filter` or `prioritize` requests in a row fail: a time-out, a connection error or an invalid response. For 30 seconds it sends no `filter` and `prioritize` requests to this extender, then it sends one probe request. If the probe succeeds, the requests resume; if it fails, the pause starts again. A response with an error in its body is not a failure: the extender has answered. During the pause, Pods are scheduled without the filter and the scores of this extender. `bind` and `preempt` requests are always sent, and extenders with `failurePolicy: Fail` are never paused.

An extender that scores nodes by their load can receive the Pods that kube-scheduler has just placed. To enable it, set `sendPlacements: true` in KubeSchedulerWebhookConfiguration. kube-scheduler then adds these Pods to the next `prioritize` request, and the extender takes them into account when it scores a burst of Pods.

How to check which node a Pod would be scheduled to

kube-scheduler can simulate the scheduling of a Pod. It evaluates the Pod with the same plugins and extenders as in a real scheduling cycle, but does not place the Pod and does not change the cluster. The answer lists every node: whether the Pod fits it, which filter or extender rejected the node and why, and the score that each plugin and extender gave the node.

The simulation is available at `POST /scheduler/simulate` of the Service `control-plane-proxy` in the `kube-system` namespace (port `8080`, HTTPS). Access is granted by the ClusterRole `d8:control-plane-manager:scheduler-simulate`. Deckhouse Platform does not bind this ClusterRole to users: bind it to the users or ServiceAccounts that call the simulation.

The request body contains the following fields:

- `namespace` and `name`: Namespace and name of an existing Pod to simulate. Instead of them, you can pass the full Pod manifest in the `pod` field.
- `excludeNodes`: Names of the nodes to skip, for example, the node that the Pod is about to leave. Optional.
- `removePods`: Pods to treat as already deleted, as objects with the `namespace`, `name` and optional `uid` fields. Optional.

The request accepts the following query parameters:

- `weights=true`: Show the weight behind every score.
- `filters=all`: List every filter that a node fails. Without the parameter, the answer contains the first filter only. A simulation with this parameter takes longer; use it for debugging.

To simulate the scheduling of a Pod, run the following commands from a Pod in the cluster:

```
d8 k create token <SERVICE_ACCOUNT> -n <SERVICE_ACCOUNT_NAMESPACE> > token
curl -sk \
  -X POST \
  -H "Authorization: Bearer $(cat token)" \
  -H "Content-Type: application/json" \
  -d '{"namespace": "<POD_NAMESPACE>", "name": "<POD_NAME>"}' \
  "https://control-plane-proxy.kube-system.svc:8080/scheduler/simulate?weights=true"
```

In the commands:

- `<SERVICE_ACCOUNT>` and `<SERVICE_ACCOUNT_NAMESPACE>`: ServiceAccount that is bound to the ClusterRole `d8:control-plane-manager:scheduler-simulate`, and its namespace.
- `<POD_NAMESPACE>` and `<POD_NAME>`: Pod to simulate.
- `-k`: Skips the verification of the `control-plane-proxy` certificate.

The response contains the `bestNodes` list with the nodes that the Pod would get now. The `nodes` list contains the verdict for each node (`Feasible`, `Rejected` or `NotEvaluated`), the rejection reasons, and the scores of plugins (`plugin/<name>`) and extenders (`extender/<name>`). The `leader` field shows whether the answer came from the kube-scheduler leader: another replica may not yet know about the Pods that the leader has just placed.

⚠ While a simulation runs, real scheduling on the same kube-scheduler replica waits for it, extender calls included (up to 30 seconds). A replica runs one simulation at a time and answers `429` to a concurrent request. Do not call the simulation in a loop without a pause.

Extenders receive the `filter` and `prioritize` requests of a simulation with the `"dryRun": true` field. On such a request, an extender must compute its answer without side effects, such as reservations. An extender that rejects unknown JSON fields fails the requests of a simulation; its requests of real scheduling are not affected.

The `descheduler` module uses the simulation in the `LowNodeUtilization` strategy: it evicts a Pod only if kube-scheduler would place it on an underutilized node.

How to monitor the scheduler

The “Scheduling” dashboard in the “Kubernetes Cluster” folder of Grafana shows how kube-scheduler works: scheduling attempts and their results, the plugin or extender that decided the node of a Pod, the number of feasible nodes, requests to extenders (duration, errors, paused extenders) and requests to the simulation.

kube-scheduler exports the following metrics:

- `scheduler_decision_deciding_plugin_total{profile, plugin}`: What decided the node of a Pod. `plugin` is `plugin/<name>` (a score plugin), `extender/<name>` (an extender), `scheduler/single-node` (only one node fit), `scheduler/tie` (the best nodes have equal scores), `scheduler/nominated-node` or `scheduler/node-hint`.
- `scheduler_extender_request_duration_seconds{extender, verb, result}`: Requests to extenders and their results: `success`, `extender_error`, `timeout`, `error`, `status`, `decode`, `skipped`. `extender` is the name of the KubeSchedulerWebhookConfiguration, with `/<n>` if it has several webhooks.
- `scheduler_extender_circuit_open{extender}`: `1` if the extender is paused now.
- `scheduler_simulate_requests_total{code}` and `scheduler_simulate_duration_seconds`: Requests to the simulation and the time that real scheduling waited for them.

How does kubelet certificate rotation work?

In Deckhouse Platform, kubelet certificate rotation is automatic.

The kubelet uses a client TLS certificate (`/var/lib/kubelet/pki/kubelet-client-current.pem`) with which it can request a new client certificate or a new server certificate (`/var/lib/kubelet/pki/kubelet-server-current.pem`) from kube-apiserver.

When there is 5-10% (random value from the range) of time left before the certificate expires, kubelet requests a new certificate from kube-apiserver. For a description of the algorithm, see the official Kubernetes documentation.

Certificates lifetime

By default, lifetime of certificates is 1 year (8760 hours).

⚠ If the client certificate lifetime has expired, kubelet will not be able to make requests to kube-apiserver and will not be able to renew certificates. In this case, the node will be marked as `NotReady` and recreated.

Specifics of working with kubelet server certificates in Deckhouse Platform

Deckhouse Platform uses IP addresses for kubelet API requests. The kubelet configuration does not use the `tlsCertFile` and `tlsPrivateKeyFile` fields, but uses a dynamic certificate that kubelet generates itself. Also, the CIS benchmark `AVD-KCV-0088` and `AVD-KCV-0089` checks, which track whether the `--tls-cert-file` and `--tls-private-key-file` arguments were passed to kubelet, are disabled in the Deckhouse Platform (in the `operator-trivy` module).

Information about the logic of working with server certificates in Kubernetes

Kubelet handles server certificates using the following logic:

- If `tlsCertFile` and `tlsPrivateKeyFile` are not empty, kubelet will use them as the default certificate and key.
  - When a client requests the kubelet API by specifying an IP address (e.g., `https://10.1.1.2:10250/`), the default private key (`tlsPrivateKeyFile`) will be used to establish a TLS connection. In this case, certificate rotation will not work.

    - When a client requests the kubelet API by specifying a host name (e.g., `https://k8s-node:10250/`), a dynamically generated private key from the `/var/lib/kubelet/pki/` directory will be used to establish a TLS connection. In this case, certificate rotation will work.
- If `tlsCertFile` and `tlsPrivateKeyFile` are empty, a dynamically generated private key from the `/var/lib/kubelet/pki/` directory will be used to establish the TLS connection. In this case, certificate rotation will work.

How to manually update control plane component certificates?

There may be a situation when the cluster's master nodes are powered off for an extended period. During this time, the control plane component certificates may expire.

Automatic renewal (normal operation): `control-plane-manager` monitors certificate expiry and automatically renews control-plane certificates when they are within 30 days of their expiry date. No manual action is required while the cluster is running.

When nodes come back online after extended downtime: Once the master nodes start and the Kubernetes API becomes available, `control-plane-manager` detects expired or soon-to-expire certificates and creates renewal operations automatically. To confirm that renewal has completed, check that the cert-renewal `ControlPlaneOperation` objects show `Phase=OperationCompleted`:

```
d8 k get cpo -o wide
```

The `ControlPlaneNode` object's `CERTIFICATES` column also shows `True` once all certificates are healthy:

```
d8 k get cpn
```

How do I protect sensitive fields in custom resources?

To protect sensitive fields (such as passwords, tokens, or keys) in resource schemas from unauthorized access via the API, unencrypted storage in etcd, or exposure in audit logs, use the `CRDSensitiveData` feature gate together with the `x-kubernetes-sensitive-data` schema marker.

To enable field protection, do the following:

1. Enable etcd encryption using the `apiserver.encryptionEnabled` parameter in the module settings. The `CRDSensitiveData` feature gate is enabled by default and shouldn't be specified manually.

   ⚠ Enabling `apiserver.encryptionEnabled` is irreversible and triggers a `kube-apiserver` restart.

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 3
     enabled: true
     settings:
       apiserver:
         encryptionEnabled: true
   ```

2. Mark sensitive fields in the resource schema with `x-kubernetes-sensitive-data: true`.

   The marker can be applied to the fields of the following types: `string`, `integer`, `number`, `boolean`, `object` and `array` (applying the marker to an `object` or `array` makes the entire subtree sensitive). Fields marked with `x-kubernetes-int-or-string: true` are supported as well.

   The marker can't be set on the schema root (the `openAPIV3Schema` node) and inside `anyOf`, `oneOf`, `allOf`, or `not` branches.

3. For users and ServiceAccounts that require access to full data, grant RBAC permissions for the `<resource>/sensitive` subresource.

Protection mechanisms

Protection	Description
Encryption in etcd	The entire resource is encrypted using the same AES-CBC transformer as Kubernetes Secrets.
API field filtering	Sensitive fields are removed from responses to `get`, `list`, and `watch` requests if the caller does not have permissions on the `<resource>/sensitive` subresource.
Audit log masking	Values of sensitive fields are replaced with `"******"` in all audit events, regardless of RBAC permissions and audit level.

A complete configuration example and results are available in the Examples section.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Built-in audit rules

> The module lifecycle stage: General Availability

What is Audit Policy

Audit Policy is a YAML file that defines which events the API server should record. When a request occurs, Kubernetes checks it against the rules in the file. The **first matching rule** determines the logging level.

Policy structure

Kubernetes documentation Policy resource field structure

In Kubernetes, an audit policy is defined as a YAML file and consists of a set of rules that determine which events are logged and at what level of detail. The file has the following structure:

```
apiVersion: audit.k8s.io/v1  # API version used for the audit policy
kind: Policy              # Resource type — always Policy
rules:                   # Set of audit rules
  - level:                 # Log detail level. Required field
    users:                 # List of users whose actions are logged
    userGroups:             # User groups (for example, system:serviceaccounts)
    verbs:                 # Logged actions/operations (create, update, delete, etc.)
    resources:              # Kubernetes resources this rule applies to
    namespaces:             # Namespaces covered by this rule
```

The `rules` array defines audit rules. Each rule contains the following fields:

- `level` — the level of detail for logged events. Possible values (from most detailed to least detailed):
  - `None` — do not log,
  - `Metadata` — only request metadata (who, when, what, from where; without object content),
  - `Request` — also stores the request body (for write operations),
  - `RequestResponse` — stores both request and response bodies.
- `users` — a list of user names the rule applies to (for example, `["admin"]`). If these are service accounts, the name usually looks like `system:serviceaccount:<namespace>:<serviceaccount-name>`. For regular users, the name depends on the authentication system configuration. For `deckhouse.io/v1` objects, `Users` uses `email` as the name.
- `userGroups` — user groups (for example, `["system:authenticated"]`). After authentication, kube-apiserver assigns each user a list of groups (for example, all authenticated users are part of `system:authenticated`, service accounts belong to additional groups). If a request comes from a user who belongs to at least one group listed in `userGroups`, the rule is applied to that request.

  Built-in Kubernetes groups:

  - `system:authenticated` — all authenticated users.
  - `system:unauthenticated` — requests from anonymous users.
  - `system:serviceaccounts` — all service accounts in all namespaces.
  - `system:serviceaccounts:<namespace>` — service accounts in a specific namespace.
- `verbs` — list of API operations (`get`, `list`, `create`, `delete`, etc.).
- `resources` — array of target resources:
  - `group` — API group (for example, `"apps"`, `"batch"`, `""` for core),
  - `resources` — resource kinds (for example, `["pods", "deployments"]`). You can get the full list of resources and groups with `kubectl api-resources`.
- `namespaces` — array of namespaces where the rule applies.
- `nonResourceURLs` —— set of URL paths to audit. The `*` symbol is allowed only as a full, final path segment. Examples:
  - `/metrics` — log requests to apiserver metrics,
  - `/healthz*` — log all health check requests.

Built-in audit rules

Deckhouse Platform deploys a basic audit policy by default, which can be extended with user-defined rules.

The policy implements the following rules:

Do not log health probes (`/readyz`, `/livez`, `/healthz`). This rule does not apply to requests that fail authentication.

```
level: None
userGroups:
  - system:authenticated
  - system:unauthenticated
nonResourceURLs:
  - /readyz
  - /readyz/*
  - /livez
  - /livez/*
  - /healthz
  - /healthz/*
```

Do not log frequent updates for `Endpoints`, `EndpointSlices`, and `Events`.

```
level: None
resources:
  - resources:
      - endpoints
      - endpointslices
      - events
```

Do not log leader election operations on `Lease` resources.

```
level: None
resources:
  - group: coordination.k8s.io
    resources:
      - leases
```

Do not log cert-manager leader election ConfigMaps.

```
level: None
resources:
  - resources:
      - configmaps
    resourceNames:
      - cert-manager-cainjector-leader-election
      - cert-manager-controller
```

Do not log VerticalPodAutoscalerCheckpoints resources.

```
level: None
resources:
  - group: autoscaling.k8s.io
    resources:
      - verticalpodautoscalercheckpoints
```

Do not log PATCH operations on `VerticalPodAutoscaler` from recommender.

```
level: None
users:
  - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-recommender
verbs:
  - patch
resources:
  - group: autoscaling.k8s.io
    resources:
      - verticalpodautoscalers
```

Do not log UpmeterHookProbes resources.

```
level: None
resources:
  - group: deckhouse.io
    resources:
      - upmeterhookprobes
```

Do not log any operations in `d8-upmeter` namespace.

```
level: None
namespaces:
  - d8-upmeter
```

Do not log ingress-nginx leader election updates in ConfigMaps.

```
level: None
users:
  - system:serviceaccount:d8-ingress-nginx:ingress-nginx
verbs:
  - update
resources:
  - resources:
     - configmaps
namespaces:
  - d8-ingress-nginx
```

Do not log dex health-check create/delete operations on `AuthRequest` resources.

```
level: None
users:
  - system:serviceaccount:d8-user-authn:dex
verbs:
  - create
  - delete
resources:
  - group: dex.coreos.com
    resources:
     - authrequests
namespaces:
  - d8-user-authn
```

Log create and delete operations for Node resources with request/response payload.

```
level: RequestResponse
verbs:
  - create
  - delete
resources:
  - resources:
     - nodes
```

Log kubectl logs requests (pods/log) at Metadata level.

```
level: Metadata
resources:
  - resources:
     - pods/log
```

Log create/update/patch/delete operations from system service accounts (`kube-system`, `d8-*`).

```
level: Metadata
users:
  - system:serviceaccount:d8-cert-manager:cainjector
  - system:serviceaccount:d8-cert-manager:cert-manager
  - system:serviceaccount:d8-cert-manager:webhook
  - system:serviceaccount:d8-cert-manager:yandex-dns-webhook
  - system:serviceaccount:d8-chrony:chrony-exporter
  - system:serviceaccount:d8-chrony:chrony-exporter-master
  - system:serviceaccount:d8-cloud-instance-manager:caps-controller-manager
  - system:serviceaccount:d8-cloud-instance-manager:cluster-autoscaler
  - system:serviceaccount:d8-cloud-instance-manager:fencing-agent
  - system:serviceaccount:d8-cloud-instance-manager:fencing-controller
  - system:serviceaccount:d8-cloud-instance-manager:integrity-containerd-configurator
  - system:serviceaccount:d8-cloud-instance-manager:integrity-controller
  - system:serviceaccount:d8-cloud-instance-manager:machine-controller-manager
  - system:serviceaccount:d8-cloud-instance-manager:node-controller
  - system:serviceaccount:d8-cloud-instance-manager:node-group
  - system:serviceaccount:d8-cloud-instance-manager:node-group-exporter
  - system:serviceaccount:d8-cloud-instance-manager:registry-packages-proxy
  - system:serviceaccount:d8-cloud-provider-aws:cloud-controller-manager
  - system:serviceaccount:d8-cloud-provider-aws:cloud-data-discoverer
  - system:serviceaccount:d8-cloud-provider-aws:node-termination-handler
  - system:serviceaccount:d8-cloud-provider-azure:cloud-controller-manager
  - system:serviceaccount:d8-cloud-provider-azure:cloud-data-discoverer
  - system:serviceaccount:d8-cloud-provider-dynamix:capd-controller-manager
  - system:serviceaccount:d8-cloud-provider-dynamix:cloud-controller-manager
  - system:serviceaccount:d8-cloud-provider-dynamix:cloud-data-discoverer
  - system:serviceaccount:d8-cloud-provider-gcp:cloud-controller-manager
  - system:serviceaccount:d8-cloud-provider-gcp:cloud-data-discoverer
```

```
- system:serviceaccount:d8-cloud-provider-huaweicloud:caphc-controller-manager
- system:serviceaccount:d8-cloud-provider-huaweicloud:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-huaweicloud:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-openstack:capo-controller-manager
- system:serviceaccount:d8-cloud-provider-openstack:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-openstack:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-vcd:capcd-controller-manager
- system:serviceaccount:d8-cloud-provider-vcd:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-vcd:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-vcd:infra-controller-manager
- system:serviceaccount:d8-cloud-provider-vsphere:capv-controller-manager
- system:serviceaccount:d8-cloud-provider-vsphere:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-vsphere:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-yandex:capy-controller-manager
- system:serviceaccount:d8-cloud-provider-yandex:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-yandex:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-yandex:cloud-metrics-exporter
- system:serviceaccount:d8-cloud-provider-yandex:validation-webhook
- system:serviceaccount:d8-cloud-provider-zvirt:capz-controller-manager
- system:serviceaccount:d8-cloud-provider-zvirt:cloud-controller-manager
- system:serviceaccount:d8-cloud-provider-zvirt:cloud-data-discoverer
- system:serviceaccount:d8-cloud-provider-zvirt:validation-webhook
- system:serviceaccount:d8-cni-cilium:agent
- system:serviceaccount:d8-cni-cilium:clustermesh-apiserver
- system:serviceaccount:d8-cni-cilium:clustermesh-controller
- system:serviceaccount:d8-cni-cilium:egress-gateway-agent
- system:serviceaccount:d8-cni-cilium:operator
- system:serviceaccount:d8-cni-cilium:relay
- system:serviceaccount:d8-cni-cilium:safe-agent-updater
- system:serviceaccount:d8-cni-cilium:ui
- system:serviceaccount:d8-cni-flannel:cni-flannel
- system:serviceaccount:d8-cni-simple-bridge:cni-simple-bridge
- system:serviceaccount:d8-csi-vsphere:cloud-data-discoverer
- system:serviceaccount:d8-descheduler:descheduler
- system:serviceaccount:d8-infra-provider-baremetal:baremetal-instance-manager
- system:serviceaccount:d8-infra-provider-baremetal:baremetal-operator
- system:serviceaccount:d8-infra-provider-baremetal:capm3
- system:serviceaccount:d8-infra-provider-baremetal:cloud-controller-manager
- system:serviceaccount:d8-infra-provider-baremetal:ironic-standalone-operator
- system:serviceaccount:d8-istio:alliance-ambientgateway
- system:serviceaccount:d8-istio:alliance-healthcheck
- system:serviceaccount:d8-istio:alliance-ingressgateway
- system:serviceaccount:d8-istio:alliance-metadata-exporter
- system:serviceaccount:d8-istio:cni
- system:serviceaccount:d8-istio:config-analyzer
- system:serviceaccount:d8-istio:ingress-gateway-controller
- system:serviceaccount:d8-istio:kiali
- system:serviceaccount:d8-istio:multicluster-api-proxy
- system:serviceaccount:d8-istio:multicluster-metrics-exporter
- system:serviceaccount:d8-istio:waypoint-controller
- system:serviceaccount:d8-istio:ztunnel
- system:serviceaccount:d8-metallb:controller
- system:serviceaccount:d8-metallb:l2lb-controller
- system:serviceaccount:d8-metallb:l2lb-speaker
- system:serviceaccount:d8-metallb:speaker
- system:serviceaccount:d8-multitenancy-manager:multitenancy-manager
- system:serviceaccount:d8-openvpn:openvpn
- system:serviceaccount:d8-system:deckhouse
- system:serviceaccount:d8-system:documentation
- system:serviceaccount:d8-system:network-policy-engine
- system:serviceaccount:d8-system:registry-controller
- system:serviceaccount:d8-system:registry-storage
- system:serviceaccount:d8-system:terraform-auto-converger
- system:serviceaccount:d8-system:terraform-state-exporter
- system:serviceaccount:d8-system:webhook-handler
- system:serviceaccount:d8-user-authn:basic-auth-proxy
- system:serviceaccount:d8-user-authn:controller
- system:serviceaccount:d8-user-authn:dex
- system:serviceaccount:d8-user-authn:dexclient-controller
- system:serviceaccount:d8-user-authn:user-api
- system:serviceaccount:d8-user-authz:controller
- system:serviceaccount:d8-user-authz:permission-browser-apiserver
- system:serviceaccount:d8-user-authz:webhook
- system:serviceaccount:kube-system:d8-control-plane-manager
- system:serviceaccount:kube-system:d8-control-plane-manager-control-plane-proxy
- system:serviceaccount:kube-system:d8-kube-dns
```

```
  - system:serviceaccount:kube-system:d8-kube-proxy
  - system:serviceaccount:kube-system:d8-node-local-dns
  - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-admission-controller
  - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-recommender
  - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-updater
  - system:serviceaccount:kube-system:node-local-dns-safe-updater
  - system:serviceaccount:kube-system:stale-dns-connections-cleaner
userGroups:
  - system:serviceaccounts
verbs:
  - create
  - update
  - patch
  - delete
omitStages:
  - RequestReceived
```

Log create/update/patch/delete operations for Pod resources.

```
level: Request
verbs:
  - create
  - delete
  - patch
  - update
resources:
  - resources:
     - pods
omitStages:
  - RequestReceived
```

Log Pod status updates from kubelet (pods/status) with the request body to capture container terminations and their reasons.

```
level: Request
userGroups:
  - system:nodes
verbs:
  - patch
  - update
resources:
  - resources:
     - pods/status
omitStages:
  - RequestReceived
```

Log create/update/patch/delete operations in system namespaces (`kube-system`, `d8-*`).

```
level: Metadata
verbs:
  - create
  - update
  - patch
  - delete
namespaces:
  - d8-admission-policy-engine
  - d8-cert-manager
  - d8-chrony
  - d8-cloud-instance-manager
  - d8-cloud-provider-aws
  - d8-cloud-provider-azure
  - d8-cloud-provider-dvp
  - d8-cloud-provider-dynamix
  - d8-cloud-provider-gcp
  - d8-cloud-provider-huaweicloud
  - d8-cloud-provider-openstack
  - d8-cloud-provider-vcd
  - d8-cloud-provider-vsphere
  - d8-cloud-provider-yandex
  - d8-cloud-provider-zvirt
  - d8-cni-cilium
  - d8-cni-flannel
  - d8-cni-simple-bridge
  - d8-csi-vsphere
  - d8-descheduler
  - d8-infra-provider-baremetal
  - d8-istio
  - d8-keepalived
  - d8-local-path-provisioner
  - d8-metallb
  - d8-monitoring
  - d8-multitenancy-manager
  - d8-network-gateway
  - d8-okmeter
  - d8-openvpn
  - d8-system
  - d8-user-authn
  - d8-user-authz
  - kube-system
omitStages:
  - RequestReceived
```

Log all LIST operations in all namespaces.

```
level: Metadata
verbs:
  - list
```

Log create and delete operations for ServiceAccount resources.

```
level: Metadata
verbs:
  - create
  - delete
resources:
  - resources:
     - serviceaccounts
omitStages:
  - RequestReceived
```

Log create/update/delete/patch operations for Role and ClusterRole resources.

```
level: Request
verbs:
  - create
  - update
  - delete
  - patch
resources:
  - group: rbac.authorization.k8s.io
    resources:
      - roles
      - clusterroles
omitStages:
  - RequestReceived
```

Log create/update/delete operations for ClusterRoleBinding resources.

```
level: Request
verbs:
  - create
  - update
  - delete
resources:
  - group: rbac.authorization.k8s.io
    resources:
      - clusterrolebindings
omitStages:
  - RequestReceived
```

Log exec, attach and ephemeral container related pod subresource operations.

```
level: Request
verbs:
  - get
  - patch
  - create
resources:
  - resources:
      - pods/exec
      - pods/attach
      - pods/ephemeralcontainers
omitStages:
  - RequestReceived
```

Log creation of VirtualMachineOperation resources with request/response payload.

```
level: RequestResponse
verbs:
  - create
resources:
  - group: virtualization.deckhouse.io
    resources:
      - virtualmachineoperations
```

Log create/update/patch/delete operations for virtualization.deckhouse.io resources.

```
level: Metadata
verbs:
  - create
  - update
  - patch
  - delete
resources:
  - group: virtualization.deckhouse.io
```

Log update/patch operations for internal virtualization subresources.

```
level: Metadata
verbs:
  - update
  - patch
resources:
  - group: internal.virtualization.deckhouse.io
    resources:
      - internalvirtualizationvirtualmachineinstances
```

Log GET operations for subresources.virtualization.deckhouse.io API group.

```
level: Metadata
verbs:
  - get
resources:
  - group: subresources.virtualization.deckhouse.io
```

Log create/update/patch/delete operations for `Pod` resources.

```
level: Metadata
verbs:
  - create
  - update
  - patch
  - delete
resources:
  - resources:
      - pods
```

Log create/update/patch/delete operations in `d8-virtualization` namespace.

```
level: Metadata
verbs:
  - create
  - update
  - patch
  - delete
namespaces:
  - d8-virtualization
```

Log create/update/patch/delete operations for ModuleConfig resources.

```
level: Metadata
verbs:
  - create
  - update
  - patch
  - delete
resources:
  - group: deckhouse.io
    resources:
      - moduleconfigs
```

Do not log requests from authenticated users.

```
level: None
userGroups:
  - system:authenticated
```

Log all remaining (unauthenticated) requests at Metadata level.

```
level: Metadata
```

Full policy example

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  # 1: Do not log health probes (`/readyz`, `/livez`, `/healthz`). This rule does not apply to requests
that fail authentication.
  - level: None
    userGroups:
      - system:authenticated
      - system:unauthenticated
    nonResourceURLs:
      - /readyz
      - /readyz/*
      - /livez
      - /livez/*
      - /healthz
      - /healthz/*

  # 2: Do not log frequent updates for `Endpoints`, `EndpointSlices`, and `Events`.
```

```
- level: None
  resources:
    - resources:
        - endpoints
        - endpointslices
        - events

# 3: Do not log leader election operations on `Lease` resources.
- level: None
  resources:
    - group: coordination.k8s.io
      resources:
        - leases

# 4: Do not log cert-manager leader election ConfigMaps.
- level: None
  resources:
    - resources:
        - configmaps
      resourceNames:
        - cert-manager-cainjector-leader-election
        - cert-manager-controller

# 5: Do not log VerticalPodAutoscalerCheckpoints resources.
- level: None
  resources:
    - group: autoscaling.k8s.io
      resources:
        - verticalpodautoscalercheckpoints

# 6: Do not log PATCH operations on `VerticalPodAutoscaler` from recommender.
- level: None
  users:
    - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-recommender
  verbs:
    - patch
  resources:
    - group: autoscaling.k8s.io
      resources:
        - verticalpodautoscalers

# 7: Do not log UpmeterHookProbes resources.
- level: None
  resources:
    - group: deckhouse.io
      resources:
        - upmeterhookprobes

# 8: Do not log any operations in `d8-upmeter` namespace.
- level: None
  namespaces:
    - d8-upmeter

# 9: Do not log ingress-nginx leader election updates in ConfigMaps.
- level: None
  users:
    - system:serviceaccount:d8-ingress-nginx:ingress-nginx
  verbs:
    - update
  resources:
    - resources:
        - configmaps
  namespaces:
    - d8-ingress-nginx

# 10: Do not log dex health-check create/delete operations on `AuthRequest` resources.
- level: None
  users:
    - system:serviceaccount:d8-user-authn:dex
  verbs:
    - create
    - delete
  resources:
    - group: dex.coreos.com
      resources:
        - authrequests
  namespaces:
    - d8-user-authn
```

```
# 11: Log create and delete operations for Node resources with request/response payload.
- level: RequestResponse
  verbs:
    - create
    - delete
  resources:
    - resources:
        - nodes

# 12: Log kubectl logs requests (pods/log) at Metadata level.
- level: Metadata
  resources:
    - resources:
        - pods/log

# 13: Log create/update/patch/delete operations from system service accounts (`kube-system`, `d8-*`).
- level: Metadata
  users:
    - system:serviceaccount:d8-cert-manager:cainjector
    - system:serviceaccount:d8-cert-manager:cert-manager
    - system:serviceaccount:d8-cert-manager:webhook
    - system:serviceaccount:d8-cert-manager:yandex-dns-webhook
    - system:serviceaccount:d8-chrony:chrony-exporter
    - system:serviceaccount:d8-chrony:chrony-exporter-master
    - system:serviceaccount:d8-cloud-instance-manager:caps-controller-manager
    - system:serviceaccount:d8-cloud-instance-manager:cluster-autoscaler
    - system:serviceaccount:d8-cloud-instance-manager:fencing-agent
    - system:serviceaccount:d8-cloud-instance-manager:fencing-controller
    - system:serviceaccount:d8-cloud-instance-manager:integrity-containerd-configurator
    - system:serviceaccount:d8-cloud-instance-manager:integrity-controller
    - system:serviceaccount:d8-cloud-instance-manager:machine-controller-manager
    - system:serviceaccount:d8-cloud-instance-manager:node-controller
    - system:serviceaccount:d8-cloud-instance-manager:node-group
    - system:serviceaccount:d8-cloud-instance-manager:node-group-exporter
    - system:serviceaccount:d8-cloud-instance-manager:registry-packages-proxy
    - system:serviceaccount:d8-cloud-provider-aws:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-aws:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-aws:node-termination-handler
    - system:serviceaccount:d8-cloud-provider-azure:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-azure:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-dynamix:capd-controller-manager
    - system:serviceaccount:d8-cloud-provider-dynamix:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-dynamix:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-gcp:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-gcp:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-huaweicloud:caphc-controller-manager
    - system:serviceaccount:d8-cloud-provider-huaweicloud:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-huaweicloud:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-openstack:capo-controller-manager
    - system:serviceaccount:d8-cloud-provider-openstack:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-openstack:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-vcd:capcd-controller-manager
    - system:serviceaccount:d8-cloud-provider-vcd:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-vcd:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-vcd:infra-controller-manager
    - system:serviceaccount:d8-cloud-provider-vsphere:capv-controller-manager
    - system:serviceaccount:d8-cloud-provider-vsphere:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-vsphere:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-yandex:capy-controller-manager
    - system:serviceaccount:d8-cloud-provider-yandex:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-yandex:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-yandex:cloud-metrics-exporter
    - system:serviceaccount:d8-cloud-provider-yandex:validation-webhook
    - system:serviceaccount:d8-cloud-provider-zvirt:capz-controller-manager
    - system:serviceaccount:d8-cloud-provider-zvirt:cloud-controller-manager
    - system:serviceaccount:d8-cloud-provider-zvirt:cloud-data-discoverer
    - system:serviceaccount:d8-cloud-provider-zvirt:validation-webhook
    - system:serviceaccount:d8-cni-cilium:agent
    - system:serviceaccount:d8-cni-cilium:clustermesh-apiserver
    - system:serviceaccount:d8-cni-cilium:clustermesh-controller
    - system:serviceaccount:d8-cni-cilium:egress-gateway-agent
    - system:serviceaccount:d8-cni-cilium:operator
    - system:serviceaccount:d8-cni-cilium:relay
    - system:serviceaccount:d8-cni-cilium:safe-agent-updater
    - system:serviceaccount:d8-cni-cilium:ui
    - system:serviceaccount:d8-cni-flannel:cni-flannel
```

```
    - system:serviceaccount:d8-cni-simple-bridge:cni-simple-bridge
    - system:serviceaccount:d8-csi-vsphere:cloud-data-discoverer
    - system:serviceaccount:d8-descheduler:descheduler
    - system:serviceaccount:d8-infra-provider-baremetal:baremetal-instance-manager
    - system:serviceaccount:d8-infra-provider-baremetal:baremetal-operator
    - system:serviceaccount:d8-infra-provider-baremetal:capm3
    - system:serviceaccount:d8-infra-provider-baremetal:cloud-controller-manager
    - system:serviceaccount:d8-infra-provider-baremetal:ironic-standalone-operator
    - system:serviceaccount:d8-istio:alliance-ambientgateway
    - system:serviceaccount:d8-istio:alliance-healthcheck
    - system:serviceaccount:d8-istio:alliance-ingressgateway
    - system:serviceaccount:d8-istio:alliance-metadata-exporter
    - system:serviceaccount:d8-istio:cni
    - system:serviceaccount:d8-istio:config-analyzer
    - system:serviceaccount:d8-istio:ingress-gateway-controller
    - system:serviceaccount:d8-istio:kiali
    - system:serviceaccount:d8-istio:multicluster-api-proxy
    - system:serviceaccount:d8-istio:multicluster-metrics-exporter
    - system:serviceaccount:d8-istio:waypoint-controller
    - system:serviceaccount:d8-istio:ztunnel
    - system:serviceaccount:d8-metallb:controller
    - system:serviceaccount:d8-metallb:l2lb-controller
    - system:serviceaccount:d8-metallb:l2lb-speaker
    - system:serviceaccount:d8-metallb:speaker
    - system:serviceaccount:d8-multitenancy-manager:multitenancy-manager
    - system:serviceaccount:d8-openvpn:openvpn
    - system:serviceaccount:d8-system:deckhouse
    - system:serviceaccount:d8-system:documentation
    - system:serviceaccount:d8-system:network-policy-engine
    - system:serviceaccount:d8-system:registry-controller
    - system:serviceaccount:d8-system:registry-storage
    - system:serviceaccount:d8-system:terraform-auto-converger
    - system:serviceaccount:d8-system:terraform-state-exporter
    - system:serviceaccount:d8-system:webhook-handler
    - system:serviceaccount:d8-user-authn:basic-auth-proxy
    - system:serviceaccount:d8-user-authn:controller
    - system:serviceaccount:d8-user-authn:dex
    - system:serviceaccount:d8-user-authn:dexclient-controller
    - system:serviceaccount:d8-user-authn:user-api
    - system:serviceaccount:d8-user-authz:controller
    - system:serviceaccount:d8-user-authz:permission-browser-apiserver
    - system:serviceaccount:d8-user-authz:webhook
    - system:serviceaccount:kube-system:d8-control-plane-manager
    - system:serviceaccount:kube-system:d8-control-plane-manager-control-plane-proxy
    - system:serviceaccount:kube-system:d8-kube-dns
    - system:serviceaccount:kube-system:d8-kube-proxy
    - system:serviceaccount:kube-system:d8-node-local-dns
    - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-admission-controller
    - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-recommender
    - system:serviceaccount:kube-system:d8-vertical-pod-autoscaler-updater
    - system:serviceaccount:kube-system:node-local-dns-safe-updater
    - system:serviceaccount:kube-system:stale-dns-connections-cleaner
  userGroups:
    - system:serviceaccounts
  verbs:
    - create
    - update
    - patch
    - delete
  omitStages:
    - RequestReceived

# 14: Log create/update/patch/delete operations for Pod resources.
- level: Request
  verbs:
    - create
    - delete
    - patch
    - update
  resources:
    - resources:
        - pods
  omitStages:
    - RequestReceived

# 15: Log Pod status updates from kubelet (pods/status) with the request body to capture container
```

```
terminations and their reasons.
  - level: Request
    userGroups:
      - system:nodes
    verbs:
      - patch
      - update
    resources:
      - resources:
          - pods/status
    omitStages:
      - RequestReceived

  # 16: Log create/update/patch/delete operations in system namespaces (`kube-system`, `d8-*`).
  - level: Metadata
    verbs:
      - create
      - update
      - patch
      - delete
    namespaces:
      - d8-admission-policy-engine
      - d8-cert-manager
      - d8-chrony
      - d8-cloud-instance-manager
      - d8-cloud-provider-aws
      - d8-cloud-provider-azure
      - d8-cloud-provider-dvp
      - d8-cloud-provider-dynamix
      - d8-cloud-provider-gcp
      - d8-cloud-provider-huaweicloud
      - d8-cloud-provider-openstack
      - d8-cloud-provider-vcd
      - d8-cloud-provider-vsphere
      - d8-cloud-provider-yandex
      - d8-cloud-provider-zvirt
      - d8-cni-cilium
      - d8-cni-flannel
      - d8-cni-simple-bridge
      - d8-csi-vsphere
      - d8-descheduler
      - d8-infra-provider-baremetal
      - d8-istio
      - d8-keepalived
      - d8-local-path-provisioner
      - d8-metallb
      - d8-monitoring
      - d8-multitenancy-manager
      - d8-network-gateway
      - d8-okmeter
      - d8-openvpn
      - d8-system
      - d8-user-authn
      - d8-user-authz
      - kube-system
    omitStages:
      - RequestReceived

  # 17: Log all LIST operations in all namespaces.
  - level: Metadata
    verbs:
      - list

  # 18: Log create and delete operations for ServiceAccount resources.
  - level: Metadata
    verbs:
      - create
      - delete
    resources:
      - resources:
          - serviceaccounts
    omitStages:
      - RequestReceived

  # 19: Log create/update/delete/patch operations for Role and ClusterRole resources.
  - level: Request
    verbs:
```

```
    - create
    - update
    - delete
    - patch
  resources:
    - group: rbac.authorization.k8s.io
      resources:
        - roles
        - clusterroles
  omitStages:
    - RequestReceived

# 20: Log create/update/delete operations for ClusterRoleBinding resources.
- level: Request
  verbs:
    - create
    - update
    - delete
  resources:
    - group: rbac.authorization.k8s.io
      resources:
        - clusterrolebindings
  omitStages:
    - RequestReceived

# 21: Log exec, attach and ephemeral container related pod subresource operations.
- level: Request
  verbs:
    - get
    - patch
    - create
  resources:
    - resources:
        - pods/exec
        - pods/attach
        - pods/ephemeralcontainers
  omitStages:
    - RequestReceived

# 22: Log creation of VirtualMachineOperation resources with request/response payload.
- level: RequestResponse
  verbs:
    - create
  resources:
    - group: virtualization.deckhouse.io
      resources:
        - virtualmachineoperations

# 23: Log create/update/patch/delete operations for virtualization.deckhouse.io resources.
- level: Metadata
  verbs:
    - create
    - update
    - patch
    - delete
  resources:
    - group: virtualization.deckhouse.io

# 24: Log update/patch operations for internal virtualization subresources.
- level: Metadata
  verbs:
    - update
    - patch
  resources:
    - group: internal.virtualization.deckhouse.io
      resources:
        - internalvirtualizationvirtualmachineinstances

# 25: Log GET operations for subresources.virtualization.deckhouse.io API group.
- level: Metadata
  verbs:
    - get
  resources:
    - group: subresources.virtualization.deckhouse.io

# 26: Log create/update/patch/delete operations for `Pod` resources.
- level: Metadata
  verbs:
```

```
    - create
    - update
    - patch
    - delete
  resources:
    - resources:
        - pods

# 27: Log create/update/patch/delete operations in `d8-virtualization` namespace.
- level: Metadata
  verbs:
    - create
    - update
    - patch
    - delete
  namespaces:
    - d8-virtualization

# 28: Log create/update/patch/delete operations for ModuleConfig resources.
- level: Metadata
  verbs:
    - create
    - update
    - patch
    - delete
  resources:
    - group: deckhouse.io
      resources:
        - moduleconfigs

############
# User-defined rules from secret kube-system/audit-policy are inserted here
############
# 29: Do not log requests from authenticated users.
- level: None
  userGroups:
    - system:authenticated

# 30: Log all remaining (unauthenticated) requests at Metadata level.
- level: Metadata
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

The module lifecycle stage: General Availability

The connection of an external scheduler plugin

Example of connecting an external scheduler plugin via webhook.

```
apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
  timeout: 5s
```

Protecting resources with sensitive fields

The following example demonstrates how to protect sensitive fields in resources using the `CRDSensitiveData` feature gate

and the `x-kubernetes-sensitive-data` schema marker.

For instructions on enabling this feature, see FAQ.

1. Enabling etcd encryption with the `encryptionEnabled` parameter. The `CRDSensitiveData` feature gate for `kube-apiserver` is enabled by default.

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 3
     enabled: true
     settings:
       apiserver:
         encryptionEnabled: true
   ```

2. Defining sensitive fields in the resource schema.

   Fields marked with `x-kubernetes-sensitive-data: true` are encrypted in etcd and removed from API responses for callers that do not have access to the `<resource>/sensitive` subresource.

   ```
   apiVersion: apiextensions.k8s.io/v1
   kind: CustomResourceDefinition
   metadata:
     name: dbconfigs.example.com
   spec:
     group: example.com
     scope: Namespaced
     names:
       plural: dbconfigs
       singular: dbconfig
       kind: DbConfig
     versions:
       - name: v1
         served: true
         storage: true
         schema:
           openAPIV3Schema:
             type: object
             properties:
               spec:
                 type: object
                 properties:
                   host:
                     type: string
                   username:
                     type: string
                   password:
                     type: string
                     x-kubernetes-sensitive-data: true
   ```

3. Creating a custom resource with values filled in sensitive fields.

   ```
   apiVersion: example.com/v1
   kind: DbConfig
   metadata:
     name: primary
     namespace: default
   spec:
     host: db.example.com
     username: admin
     password: s3cr3t
   ```

   Once saved, the entire object is encrypted in etcd and the `password` value is hidden in the audit log and removed from the API responses of the caller has no permissions to the `dbconfigs/sensitive` subresource.

4. Configuring access to sensitive fields using RBAC and the `<resource>/sensitive` subresource.

```
# Regular role: can read the resource, but sensitive fields are removed from responses.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
---
# Privileged role: can read full data, including sensitive fields.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-sensitive-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["example.com"]
  resources: ["dbconfigs/sensitive"]
  verbs: ["get", "list", "watch"]
```

5. A result of sensitive field protection.
   - A user with the `dbconfig-reader` role who runs `d8 k get dbconfig primary -o json` can see the resource with sensitive fields removed:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin"
       }
     }
     ```

   - A user with the `dbconfig-sensitive-reader` role can see the full data:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin",
         "password": "s3cr3t"
       }
     }
     ```

   - In audit logs, values of sensitive fields are always masked, regardless of caller permissions:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin",
         "password": "******"
       }
     }
     ```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The csi-vsphere module

Included in extensions: **Storage Integration**

The module lifecycle stage: Deprecated

The `csi-vsphere` module provides Container Storage Interface (CSI) support for VMware vSphere environments, enabling dynamic provisioning and management of persistent storage volumes in Kubernetes clusters running on vSphere infrastructure.

This module is specifically designed for **static Kubernetes clusters** (non-cloud) deployed on VMware vSphere and works independently from the cloud-provider-vsphere module. It integrates with vSphere datastores to provide persistent storage capabilities through the vSphere CSI driver.

> **i** Use the `csi-vsphere` module when the cluster cannot use the `cloud-provider-vsphere` module but still requires dynamic volume provisioning on vSphere infrastructure.

Main Features

- Provides dynamic persistent volume provisioning for static Kubernetes clusters on VMware vSphere.
- Automatically discovers vSphere datastores and creates StorageClass resources.
- Supports topology-aware volume placement using vSphere region and zone tags.
- Handles volume creation, deletion, attachment, detachment, and online resizing (Default mode).
- Supports Legacy, Default, and Migration compatibility modes for smooth driver transitions.

The module deploys the VMware vSphere CSI driver components that:

- **Automatically discover vSphere datastores**: Reads discovery data from the `d8-cloud-provider-discovery-data` secret to identify available datastores and their topology.
- **Create StorageClasses**: Automatically generates StorageClass resources for each discovered datastore, filtered by the exclude list if configured.
- **Provision persistent volumes**: Dynamically creates persistent volumes on vSphere datastores when PersistentVolumeClaims are created.
- **Support volume operations**: Handles volume creation, deletion, attachment, detachment, and online resizing (in Default mode).
- **Implement topology awareness**: Uses zone and region labels for proper volume placement according to vSphere cluster topology.

System requirements

To use the module in a cluster, the following system requirements must be met:

- All cluster nodes must be virtual machines created with vSphere tooling.
- The vSphere VM name must **exactly match** the node hostname in the Deckhouse Platform cluster.
- Each VM must have `disk.EnableUUID:TRUE` set so the platform can identify attached volumes reliably.

Architecture

The CSI driver consists of components:

1. **CSI Controller** (runs on master nodes):
   - **vsphere-csi-controller**: Main controller managing volume lifecycle operations;
   - **csi-provisioner**: Watches for PVCs and triggers volume creation;
   - **csi-attacher**: Handles volume attachment to nodes;
   - **csi-resizer**: Manages online volume expansion;
   - **vsphere-syncer**: Synchronizes metadata between Kubernetes and vSphere every 30 minutes;
   - **liveness-probe**: Monitors controller health.
2. **CSI Node** (DaemonSet on all nodes):
   - **vsphere-csi-node**: Node-level CSI driver handling volume mounting/unmounting;
   - **node-driver-registrar**: Registers the CSI driver with kubelet;
   - **liveness-probe**: Monitors node driver health.

Compatibility modes

The module supports the following operational modes (the mode is set via the `compatibilityFlag` parameter):

- **Default mode** (used by default, `compatibilityFlag` not set or empty):
  - uses the current vSphere CSI driver (`csi.vsphere.vmware.com`);
  - supports CNS (Cloud Native Storage) volumes;
  - enables online volume resizing;
  - recommended for new deployments.
- **Legacy mode** (`compatibilityFlag: "Legacy"`):
  - uses the older vSphere CSI driver (`vsphere.csi.vmware.com`);
  - works with FCD (First Class Disk) volumes only;
  - no online volume resizing support;
  - used for backward compatibility.
- **Migration mode** (`compatibilityFlag: "Migration"`):
  - runs both driver versions simultaneously,
  - facilitates migration from Legacy to Default mode,
  - allows gradual transition of workloads.

Storage discovery and provisioning

The module automatically:

1. **Discovers datastores** from the cloud provider discovery data.
2. **Generates StorageClass names** by sanitizing datastore names to meet Kubernetes DNS naming requirements.
3. **Filters excluded storage** based on storageClass.exclude configuration (supports regex patterns).
4. **Creates StorageClasses** with appropriate parameters:
   - `allowVolumeExpansion: true` (Default mode only).
   - `volumeBindingMode: WaitForFirstConsumer`: Ensures volumes are created in the same zone as the pod.
   - Topology constraints matching vSphere regions and zones.

Default StorageClass and exclusions

The module automatically creates a StorageClass for each Datastore and DatastoreCluster from the zones.

To set the default StorageClass for a cluster, use the global parameter `defaultClusterStorageClass`. To filter out unwanted StorageClass objects, specify exclusion patterns in the `storageClass.exclude` parameter.

Volume lifecycle

When a pod requests storage:

1. A PVC is created referencing a StorageClass managed by this module.
2. The CSI provisioner creates a volume on the specified vSphere datastore.
3. The CSI attacher attaches the volume to the appropriate node.
4. The CSI node driver mounts the volume into the pod.
5. The vsphere-syncer periodically synchronizes metadata to ensure consistency.

Limitations

- This module is designed exclusively for static (non-cloud) Kubernetes clusters on vSphere.
- Legacy mode does not support online volume resizing.
- Volume expansion requires the current CSI driver (`csi.vsphere.vmware.com`).
- StorageClasses are automatically managed — manual modifications may be overwritten.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

Included in extensions: **Storage Integration**

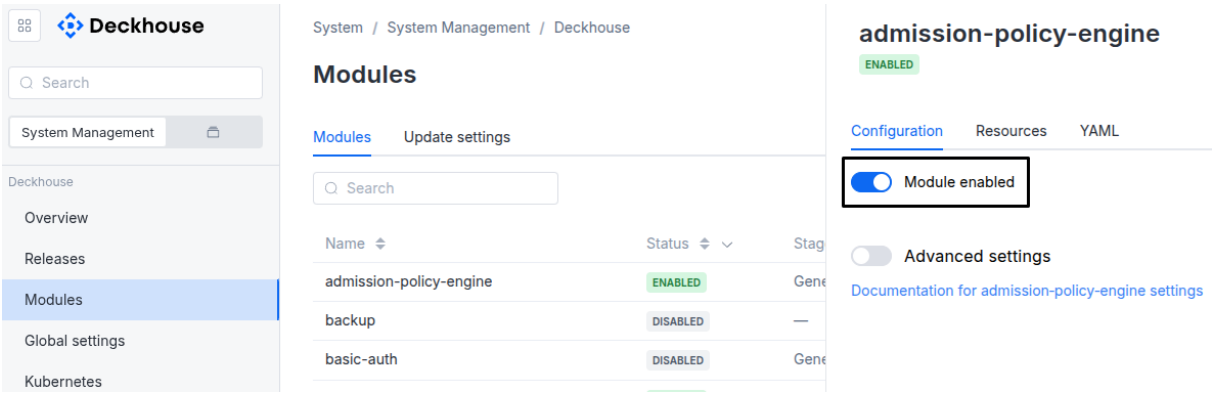
The module lifecycle stage: Deprecated

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `csi-vsphere` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable csi-vsphere
```

- **Using ModuleConfig `csi-vsphere`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `csi-vsphere` (create it if necessary);

  Example of a manifest to enable module `csi-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-vsphere
spec:
  enabled: true
```

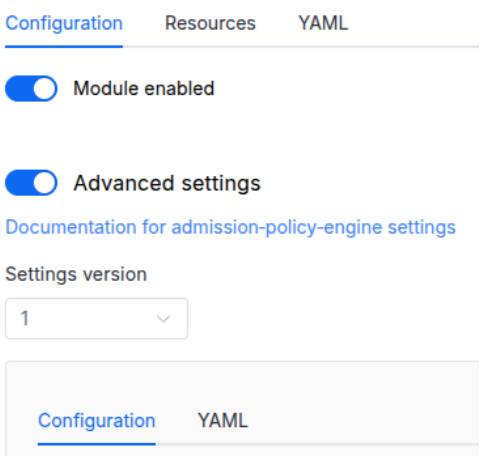
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `csi-vsphere` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `csi-vsphere` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `csi-vsphere`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `csi-vsphere` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc csi-vsphere
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `csi-vsphere` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `csi-vsphere`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: csi-vsphere
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableTimesync**
    boolean

    Disable time synchronization on the vSphere side.

    > **Warning.** This parameter does not disable NTP daemons in the guest OS; it only disables time correction on the ESXi side.

  - **settings.host**
    string

    Required value

    The domain of the vCenter server.

    Example:

    ```
    host: myhost
    ```

- **settings.insecure**
  boolean

  Set to `true` if vCenter has a self-signed certificate.

- **settings.password**
  string

  Required value

  The user's password.

  Example:

  ```
  password: myPaSsWd
  ```

- **settings.region**
  string

  Required value

  Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

  Example:

  ```
  region: myreg
  ```

- **settings.regionTagCategory**
  string

  Required value

  The name of the tag category used to identify the region (vSphere Datacenter).

  Example:

  ```
  regionTagCategory: myregtagcat
  ```

- **settings.storageClass**
  object
  - **settings.storageClass.compatibilityFlag**
    string

    A flag allowing the use of the old CSI version:

    - `Legacy` — use the old version of the driver. FCD discs only, no online-resizing;
    - `Migration` — in this case, both drivers will be available in the cluster at the same time. This mode is used to migrate from an old driver.

    Allowed values: `Legacy`, `Migration`

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-lun101-.*"
    - slow-lun103-1c280603
    ```

- **settings.username**
  string

  Required value

  The login.

  Example:

  ```
  username: myuname
  ```

- **settings.vmFolderPath**
  string

  Required value

  The path to the VirtualMachine Folder where the cloned VMs will be created.

  Example:

  ```
  vmFolderPath: dev/test
  ```

- **settings.zoneTagCategory**
  string

  Required value

The name of the tag category used to identify the zone (vSphere Cluster).

Example:

```
zoneTagCategory: myzonetagcat
```

- **settings.zones**
  array of strings

  Required value

  The globally restricted set of zones that this Cloud Provider works with.

  Example:

  ```
  zones:
  - zonea
  - zoneb
  ```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Preparing environment

The module lifecycle stage: Deprecated

This section describes how to prepare vCenter and vSphere for the `csi-vsphere` module.

Required resources

The following resources are required for the module to function:

- User with the required permissions.
- Network with DHCP and internet access.
- Datacenter tagged as `k8s-region`.
- Cluster tagged as `k8s-zone`.
- One or more Datastores with the appropriate tags.

Installing govc

Use the vSphere CLI govc.

Set environment variables for vCenter access:

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Create tags and tag categories

`csi-vsphere` maps Kubernetes topology to vSphere objects: a **region** is a vSphere Datacenter, and a **zone** is a vSphere Cluster. The relationship between these objects is defined using tags.

To link Cluster and Datacenter objects, follow these steps:

1. Create tag categories:

   ```
   govc tags.category.create -d "Kubernetes Region" k8s-region
   govc tags.category.create -d "Kubernetes Zone" k8s-zone
   ```

2. Create tags in each category. If you use several zones (Cluster), create one tag per cluster:

   ```
   govc tags.create -d "Kubernetes Region" -c k8s-region test-region
   govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
   govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
   ```

3. Attach the region tag to the Datacenter:

   ```
   govc tags.attach -C k8s-region test-region /<DatacenterName>
   ```

4. Attach zone tags to Cluster objects:

   ```
   govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
   govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
   ```

Datastore configuration

For dynamic PersistentVolume provisioning, each Datastore must be reachable from every ESXi host (shared datastore).

Attach region and zone tags to Datastore objects so the module can create StorageClass resources automatically:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Create and assign the role

Due to the variety of SSO providers connected to csi-vsphere, the steps for creating a user are not covered in this article.

The role to be created below includes all possible privileges for all DP components. For a detailed list of privileges, refer to the `cloud-provider-vsphere` documentation.

Create the role:

```
govc role.create deckhouse \
   Cns.Searchable Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
   Global.GlobalTag Global.SystemTag Network.Assign StorageProfile.View \
   $(govc role.ls Admin | grep -F -e 'Folder.' -e 'InventoryService.' -e 'Resource.' -e 'VirtualMachine.')
```

Grant the role on the `vCenter` root object:

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The deckhouse module

> Included in extensions: **Billing, Cluster Union**

> The module lifecycle stage: General Availability

In Deckhouse, this module sets up:

- **The logging level**;
- **The set of modules enabled by default**;

  Usually, the `Default` set is used (it is suitable for most cases).

  Regardless of the set of modules enabled by default, any module can be explicitly enabled or disabled in the Deckhouse configuration (learn more about enabling and disabling a module).
- **The release channel**;

  Deckhouse has a built-in mechanism for automatic updates. This mechanism uses 5 release channels with various stability and frequency of releases. Learn more about how the automatic update mechanism works and how you can set the desired release channel
- **The update mode** and **update windows**;

  Deckhouse supports **manual** and **automatic** update modes.

  In the manual upgrade mode, only critical fixes (patch releases) are automatically applied, and upgrading to a more current Deckhouse release requires manual confirmation.

  In the automatic update mode, Deckhouse switches to a newer release as soon as it is available in the corresponding release channel unless update windows are **configured** for the cluster. If update windows are **configured** for the cluster, Deckhouse will upgrade to a newer release during the next available update window.
- **Service for validating Custom Resources**.

  The validation service prevents creating Custom Resources with invalid values or adding such values to the existing Custom Resources. Note that it only tracks Custom Resources managed by Deckhouse modules.

Deckhouse releases update

Get Deckhouse releases status

You can get Deckhouse releases list via the command `d8 k get deckhousereleases`. By default, a cluster keeps the last 10 Superseded releases and all deployed/pending releases.

Every release can have one of the following statuses:

- `Pending` - release is waiting to be deployed: waiting for update window, canary deployment, etc. You can see the detailed status via the `d8 k describe deckhouserelease $name` command.
- `Deployed` - release is applied. It means that the image tag of the Deckhouse Pod was changed, but the update process of all components is going asynchronously and could not have been finished yet.
- `Superseded` - release is outdated and not used anymore.
- `Suspended` - release was suspended (for example, due to an error found). A release can have this status if it was suspended before being deployed in the cluster.

Update process

When release status is changed to `Deployed` state, release is updating only a tag of the Deckhouse image. Deckhouse will start checking and updating process of the all modules, which were changed from the last release. Duration of an update could be different and connected to cluster size, enabled modules count and settings. Foe example: if you cluster have a lot of `NodeGroup` resources, it will take some time to update them because these resources are updated one by one `IngressNginxControllers` also updating one by one.

Manual release deployment

If you have a manual update mode enabled and have a few Pending releases, you can approve them all at once. In that case Deckhouse will update in series keeping a release order and changing their status during the update.

Pin a release

To pin a release means fully or partially disable the automatic Deckhouse version update.

There are three options to limit the automatic update of Deckhouse:

- Enable a manual update mode.

  In this case, Deckhouse holds on a current version and able to receive updates. But to apply the update, a manual action will need to be performed. This applies to both patch versions and minor versions.

  To enable manual update mode, you need to set the parameter settings.update.mode in ModuleConfig `deckhouse` to `Manual`:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
  ```
- Set the automatic update mode for patch versions.

  In this case, Deckhouse holds on a current version and will automatically update to patch versions of the current release (taking into account the update windows). To apply a minor version update, a manual action will need to be performed.

  For example: the current version of DP is `v1.70.1`, after setting the automatic update mode for patch versions, Deckhouse can be updated to version `v1.70.2`, but will not update to version `v1.71.*` or higher.

  To set the automatic update mode for patch versions, you need to set the parameter settings.update.mode to `AutoPatch` in the ModuleConfig `deckhouse`:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"AutoPatch"}}}}'
  ```
- Set a specified image tag for Deployment `deckhouse` and remove releaseChannel parameter from `deckhouse` module configuration.

  In that case, DP holds on a current version, and no information about new available versions in the cluster (DeckhouseRelease objects) will be received.

  An example of installing version `v1.66.3` for DP EE and removing the `releaseChannel` parameter from the configuration of the `deckhouse` module:

  ```
  d8 k -ti -n d8-system exec svc/deckhouse-leader -c deckhouse -- kubectl set image deployment/deckhouse deckhouse=registry.deckhouse.io/deckhouse/ee:v1.66.3
  d8 k patch mc deckhouse --type=json -p='[{"op": "remove", "path": "/spec/settings/releaseChannel"}]'
  ```

Priority Classes

This module creates a set of priority classes and assigns them to components installed by Deckhouse and applications in the cluster.

Priority classes are utilized by the scheduler to determine a Pod's priority based on the class it belongs to during scheduling.

For example, when deploying Pods with `priorityClassName: production-low`, if the cluster lacks sufficient resources, Kubernetes will start evicting Pods with the lowest priority to accommodate the `production-low` Pods. That is, Kubernetes will first evict all Pods with `priorityClassName: develop` Pods, then proceed to `cluster-low` Pods, and so on.

When setting the priority class, it is crucial to understand the type of application and the environment it operates in. Assigning any priority class does not lower a Pod's priority because Pods without a specified priority class are considered to have the lowest priority.

Practical examples of creating and using PriorityClass, preemption demos, and troubleshooting are described in the Examples section.

⚠ You cannot use the following priority classes: `system-node-critical`, `system-cluster-critical`, `cluster-medium`, `cluster-low`.

Below is the list of priority classes set by the module (sorted by the priority from highest to lowest):

Priority class	Description	Value
`system-node-critical`	Cluster components that must be present on the node. This priority class fully protects components against eviction by kubelet. Examples: `node-exporter`, `csi`, and others.	2000001000
`system-cluster-critical`	Cluster components that are critical to its correct operation. This PriorityClass is mandatory for MutatingWebhooks and Extension API servers. It also fully protects components against eviction by kubelet. Examples: `kube-dns`, `kube-proxy`, `cni-flannel`, `cni-cilium`, and others.	2000000000
`production-high`	Stateful applications in the production environment. Their unavailability leads to service downtime or data loss. Examples: `PostgreSQL`, `Memcached`, `Redis`, `MongoDB`, and others.	9000
`cluster-medium`	Cluster components responsible for monitoring (alerts, diagnostic tools) and autoscaling. Without monitoring, assessing the scale of incidents is impossible; without autoscaling, applications cannot receive necessary resources. Examples: `deckhouse`, `node-local-dns`, `grafana`, `upmeter`, and others.	7000
`production-medium`	Main stateless applications in the production environment that are responsible for operating the service for end-users.	6000
`deployment-machinery`	Cluster components that are responsible for deploying/building.	5000
`production-low`	Non-critical, secondary applications in the production environment (crons, admin dashboards, batch processing). For important batch or cron jobs, consider assigning them the `production-medium` priority.	4000
`staging`	Staging environments for applications.	3000
`cluster-low`	Cluster components that are desirable but not essential for proper cluster operation. Examples: `dashboard`, `cert-manager`, `prometheus`, and others.	2000
`develop` (default)	Develop environments for applications. The default class for a component (if other priority classes aren't set).	1000
`standby`	This class is not intended for applications. It is used for system purposes (reserving nodes).	-1

Monitoring

The module provides monitoring and observability for Deckhouse Platform (DP). It collects platform performance metrics, provides pre-configured alerting rules and dashboards for DP status monitoring.

The module uses features provided by `prometheus` and `observability` modules for collecting and storing metrics.

It deploys monitoring resources that:

- **Collect DP metrics**: Collects metrics from the `deckhouse` Pod using a PodMonitor resource, including:
  - Self metrics on port `4222` via the `/metrics` endpoint.
  - Custom hook-generated metrics via the `/metrics/hooks` endpoint.
  - Module execution metrics, hook performance, and system health indicators.
- **Define alerting rules**: Provides Prometheus alerting rules organized into several categories:
  - **DP availability**: Monitors Pod health, readiness, and uptime.
  - **DP malfunctioning**: Detects excessive restarts, registry access issues, and hung processes.
  - **Release management**: Tracks release channel subscriptions, pending updates, and manual approvals.
  - **Module management**: Monitors module state, validation errors, and deprecated configurations.
  - **CNI checks**: Detects multiple CNI configurations and misconfigurations.
  - **OS requirements**: Identifies nodes running deprecated operating system versions.
- **Provide Grafana dashboards**: Includes pre-built Grafana dashboards for visualizing:
  - DP performance metrics.
  - Module execution statistics.
  - Hook run times and resource usage.
  - Queue processing and convergence status.

Metrics collection

A PodMonitor scrapes two endpoints from the `deckhouse` Pod:

1. **DP metrics** (`/metrics`): Core DP operational metrics, including the following:
   - `deckhouse_live_ticks`: Health indicator incrementing every 10 seconds.
   - `deckhouse_registry_check_errors_total`: Failed registry checks.
   - `deckhouse_module_hook_run_seconds`: Module hook execution duration.
   - `deckhouse_tasks_queue_action_duration_seconds`: Task queue processing times.
   - Other operational metrics.
2. **Hook metrics** (`/metrics/hooks`): Custom metrics generated by DP hooks with `honorLabels: true` to preserve hook-specific labels.

The PodMonitor is created only when the `operator-prometheus` module is enabled, and the alerting rules and dashboards are processed by the `prometheus` module.

Integration with the observability module

When the `observability` module is enabled, the following resources are created automatically:

- ClusterObservabilityMetricsRulesGroup: For Prometheus rules.
- ClusterObservabilityDashboard: For Grafana dashboards.

This enables centralized monitoring resource management via the `observability` module interfaces.

Third-party components

List of third-party software used in the `deckhouse` module:

- Shell operator 1.9.3

  License: Apache License 2.0

  A tool for running event-driven scripts in a Kubernetes cluster

- JQ 1.7.1

  License: Mixed (https://github.com/stedolan/jq/blob/master/COPYING)

  A lightweight and flexible command-line JSON processor.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Included in extensions: **Billing, Cluster Union**

> The module lifecycle stage: General Availability

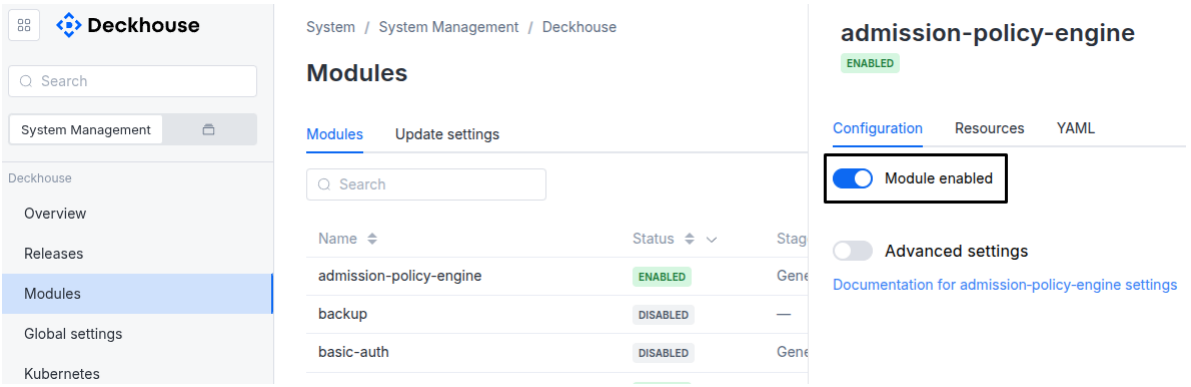
The module has 67 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable deckhouse
```

- **Using ModuleConfig `deckhouse`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `deckhouse` (create it if necessary);

  Example of a manifest to enable module `deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
```

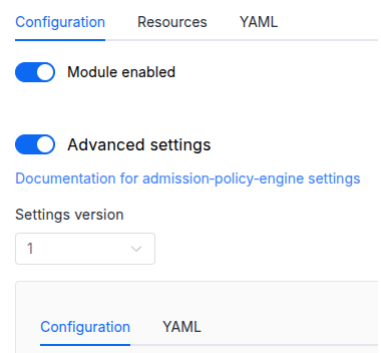
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `deckhouse` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `deckhouse`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `deckhouse` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc deckhouse
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `deckhouse` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `deckhouse`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  The `enableModuleV2` parameter is removed. Module v2 is enabled with the `moduleV2Enabled` global parameter.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.allowExperimentalModules**
    boolean

    Allows the use of experimental modules (modules in the `Experimental` lifecycle stage).

    Experimental modules can be unstable and are not supported under the Deckhouse SLA. Use them with caution.

    Default: `false`

    Example:

    ```
    allowExperimentalModules: false
    ```

  - **settings.allowedExperimentalModules**
    array of strings

    A list of experimental module names allowed to be enabled even when `allowExperimentalModules` is `false`.

    Lets you opt into specific experimental modules without allowing all of them.

    If `allowExperimentalModules` is `true`, all experimental modules are allowed and this list is ignored.

    Default: `[]`

    Example:

    ```
    allowedExperimentalModules:
    - r-b-test
    ```

  - **settings.bundle**
    string

    The set of DP modules that are enabled by default.

    Changing the parameter value in an already deployed cluster is not possible.

    Possible values:

    - `Default`: Recommended set of modules for cluster operation. Includes modules for monitoring, authorization control, networking, and other basic platform needs.
    - `Managed`: Set of modules for managed clusters from cloud providers.
    - `Minimal`: Minimum possible set, which includes a single module (this one).

      Warning. Several basic platform modules (for example, the CNI module) are not included in the `Minimal` module set. Without the basic modules, DP can only operate in an already deployed cluster. Read the “Things to keep in mind when working with the Minimal module set” section for more details.

    Default: `Default`

    Allowed values: `Default`, `Minimal`, `Managed`

    Example:

    ```
    bundle: Default
    ```

  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Example:

    ```
    highAvailability: true
    ```

  - **settings.license**
    object

Edition settings of the Deckhouse Platform.

- **settings.license.edition**
  string

  Edition of the Deckhouse Platform:

  - `CE` — Community Edition.
  - `BE` - Basic edition.
  - `EE` — Enterprise Edition.
  - `SE` — Standard Edition.
  - `SE-plus` — Standard Edition+.
  - `CSE` — Certified Security Edition.

  Default: `CE`

  Example:

  ```
  edition: CE
  ```

- **settings.logLevel**
  string

  Deckhouse logging level.

  Default: `Info`

  Allowed values: `Trace`, `Debug`, `Info`, `Error`

  Example:

  ```
  logLevel: Info
  ```

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, `nodeSelector` will be determined automatically.

  Caution! Deckhouse will stop working if there is a nonexistent label in `nodeSelector`. You need to change the values to the correct ones in `ModuleConfig/deckhouse` and `deployment/deckhouse` to get Deckhouse back on track.

- **settings.registry**
  object

  Settings for accessing the container registry with Deckhouse images.

  This parameter allows configuring how DP connects to the container registry to pull its own component images. Settings are processed by the internal registry module. In clusters where this configuration is not supported, the registry connection setup is done manually using the `change-registry` helper script.

  Examples:

  ```
  registry:
    mode: Unmanaged
  ```

  ```
  registry:
    mode: Unmanaged
    unmanaged:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  registry:
    mode: Direct
    direct:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
  ```

  - **settings.registry.direct**
    object

    Parameters for the `Direct` access mode. These parameters are required when `mode` is set to `Direct`.

    Examples:

    ```
    direct:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
    ```

    ```
    direct:
      imagesRepo: my-private-registry.com/deckhouse
      username: user
      password: password
      scheme: HTTPS
      ca: |
        -----BEGIN CERTIFICATE-----
        MIIC...
        -----END CERTIFICATE-----
    ```

    - **settings.registry.direct.ca**
      string

      The root CA certificate (in PEM format) for validating the container registry's server certificate. This is necessary if the registry uses a self-signed certificate or a certificate issued by a private Certificate Authority. If not provided, the system's trusted root CAs will be used.

    - **settings.registry.direct.checkMode**
      string

      Mode for checking the container registry:

      - `Default` — checks for the presence of all critical components in the container registry.
      - `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

      Default: `Default`

      Allowed values: `Default`, `Relax`

    - **settings.registry.direct.imagesRepo**
      string

      Required value

The address of the container registry repository where Deckhouse images are stored.

In firewalled environments, this might be a private registry address that mirrors the public one.

Default: `registry.deckhouse.io/deckhouse/ce`

Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

Examples:

```
imagesRepo: registry.deckhouse.io/deckhouse/ee
```

```
imagesRepo: private-registry:5000/deckhouse/ce
```

- **settings.registry.direct.license**
  string

  The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is mutually exclusive with `username`/`password`.

  Example:

  ```
  license: DECKHOUSE_LICENSE_KEY
  ```

- **settings.registry.direct.password**
  string

  The password for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  password: registry-password
  ```

- **settings.registry.direct.scheme**
  string

  The protocol to use for connecting to the registry.

  Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

  Example:

  ```
  scheme: HTTPS
  ```

- **settings.registry.direct.username**
  string

  The username for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  username: registry-user
  ```

- **settings.registry.mode**
  string

  The mode for accessing the container registry with Deckhouse images.

  - `Unmanaged`: The default mode. In this mode, the `registry` module does not run an internal registry. To configure it, set the parameters in the `unmanaged` section. **Important.** Changing the registry in `Unmanaged` mode will result in a full restart of all DP components.
  - `Direct`: A mode for direct access to the registry. It uses a “virtual” registry address that is overwritten on the fly to the real registry address. This allows for registry changes without a full component restart. **Important.** The first switch from `Unmanaged` to `Direct` mode will result in a full restart of all DP components.
  - `Proxy` and `Local`: Deprecated and will be removed in a future release. These are modes of the v1 implementation of the `registry` module (DP up to and including 1.77), and they have no effect: the installer refuses them, and they cannot be set in a running cluster. To run an in-cluster cache or to operate an air-gapped cluster, use the `registry` module.

  Default: `Unmanaged`

  Allowed values: `Unmanaged`, `Direct`, `Proxy`, `Local`

- **settings.registry.proxy**
  Deprecated
  object

  Deprecated. Parameters for the `Proxy` mode of the v1 implementation of the `registry` module. The mode has no effect, see the `mode` parameter.

  - **settings.registry.proxy.ca**
    string

    The root CA certificate (in PEM format) for validating the container registry’s server certificate. This is necessary if the registry uses a self-signed certificate or a certificate issued by a private Certificate Authority. If not provided, the system’s trusted root CAs will be used.

  - **settings.registry.proxy.checkMode**
    string

    Mode for checking the container registry:

    - `Default` — checks for the presence of all critical components in the container registry.
    - `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

    Default: `Default`

    Allowed values: `Default`, `Relax`

  - **settings.registry.proxy.imagesRepo**
    string

    Required value

    The address of the container registry repository where Deckhouse images are stored.

    In firewalled environments, this might be a private registry address that mirrors the public one.

    Default: `registry.deckhouse.io/deckhouse/ce`

    Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

    Examples:

    ```
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    ```

    ```
    imagesRepo: private-registry:5000/deckhouse/ce
    ```

- **settings.registry.proxy.license**
  string

  The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is mutually exclusive with `username/password`.

  Example:

  ```
  license: DECKHOUSE_LICENSE_KEY
  ```

- **settings.registry.proxy.password**
  string

  The password for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  password: registry-password
  ```

- **settings.registry.proxy.scheme**
  string

  The protocol to use for connecting to the registry.

  Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

  Example:

  ```
  scheme: HTTPS
  ```

- **settings.registry.proxy.ttl**
  string

  TTL determines the lifetime of cached images. Once the TTL expires, the images will be automatically removed from the cache. If not set, the TTL defaults to 7 days (168 hours).

  Supported units: `h` (hours), `m` (minutes), `s` (seconds). The minimum allowable value is 5 minutes.

  Pattern: `^(?:(\d+)h)?(?:(\d+)m)?(?:(\d+)s)?$`

- **settings.registry.proxy.username**
  string

  The username for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  username: registry-user
  ```

- **settings.registry.unmanaged**
  object

  Parameters for the `Unmanaged` access mode.

  Examples:

  ```
  unmanaged:
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  unmanaged:
    imagesRepo: my-private-registry.com/deckhouse
    username: user
    password: password
    scheme: HTTPS
    ca: |
      -----BEGIN CERTIFICATE-----
      MIIC...
      -----END CERTIFICATE-----
  ```

  - **settings.registry.unmanaged.ca**
    string

    The root CA certificate (in PEM format) for validating the container registry's server certificate. This is necessary if the registry uses a self-signed certificate or a certificate issued by a private Certificate Authority. If not provided, the system's trusted root CAs will be used.

  - **settings.registry.unmanaged.checkMode**
    string

    Mode for checking the container registry:

    - `Default` — checks for the presence of all critical components in the container registry.
    - `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

    Default: `Default`

    Allowed values: `Default`, `Relax`

  - **settings.registry.unmanaged.imagesRepo**
    string

    Required value

    The address of the container registry repository where Deckhouse images are stored.

    In firewalled environments, this might be a private registry address that mirrors the public one.

    Default: `registry.deckhouse.io/deckhouse/ce`

    Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

    Examples:

    ```
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    ```

    ```
    imagesRepo: private-registry:5000/deckhouse/ce
    ```

  - **settings.registry.unmanaged.license**
    string

    The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is

mutually exclusive with `username`/`password`.

Example:

```
license: DECKHOUSE_LICENSE_KEY
```

- **settings.registry.unmanaged.password**
  string

  The password for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  password: registry-password
  ```

- **settings.registry.unmanaged.scheme**
  string

  The protocol to use for connecting to the registry.

  Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

  Example:

  ```
  scheme: HTTPS
  ```

- **settings.registry.unmanaged.username**
  string

  The username for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  username: registry-user
  ```

- **settings.releaseChannel**
  string

  Target Deckhouse release channel (Deckhouse will switch to it when such an opportunity appears).

  The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

  Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`

  Example:

  ```
  releaseChannel: Stable
  ```

- **settings.reservedPublicHosts**
  object

  Settings of the reservation that keeps the hostnames of the DP web interfaces from being claimed by a workload.

  DP renders those hostnames from the `global.modules.publicDomainTemplate` global parameter and reserves them cluster-wide: an Ingress, or a Gateway, GRPCRoute, HTTPRoute, ListenerSet or TLSRoute of the Gateway API, created outside a system namespace cannot name one of them. Using the settings in `reservedPublicHosts`, you can expand the set of reserved names (using the `additionalHosts` parameter) or reduce it (using the `excludedServices` parameter).

  The reservation is per hostname and applies to every namespace. To exempt a whole namespace from the reservation instead, label the namespace with `security.deckhouse.io/reserved-hosts-bypass: "true"`.

  Example:

  ```
  reservedPublicHosts:
    mode: List
    additionalHosts:
    - admin.example.com
    excludedServices:
    - grafana
  ```

  - **settings.reservedPublicHosts.additionalHosts**
    array of strings

    Hostnames to reserve on top of the ones DP publishes.

    Every entry is a complete hostname, matched exactly and case-insensitively, the same way the built-in ones are. Wildcards are not supported.

    The hostname specified in `additionalHosts` is reserved, even if the name of the corresponding service is listed in `excludedServices`.

    Default: `[]`

    Example:

    ```
    additionalHosts:
    - admin.example.com
    ```

    - **Element of the array**
      string

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **settings.reservedPublicHosts.excludedServices**
    array of strings

    Service names whose hostname must not be reserved.

    Use it for a cluster where a workload outside a system namespace legitimately serves a path under one of those hostnames.

    A name is the part `global.modules.publicDomainTemplate` substitutes, `grafana` for `grafana.example.com`, not the hostname itself. The hostname it renders to is published in the `excludedHosts` key of the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace, which is where to check that the name did what was meant.

    Under `mode: Template` any name works, because the reservation covers the whole domain and not only what Deckhouse publishes. Under `mode: List` only a name Deckhouse actually publishes has an effect, because only those are reserved there; the rest are listed in the `unknownExcludedServices` key of the same ConfigMap, which is where to look if a hostname stayed reserved unexpectedly.

    Specifying a service name in `excludedServices` prevents the corresponding hostname from being reserved in all namespaces, including those where the platform itself hosts it.

    Default: `[]`

    Example:

```
excludedServices:
- grafana
```

- **Element of the array**
  string

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.reservedPublicHosts.mode**
  string

  How much of the platform's domain is reserved.

  Set `Template` to reserve every hostname the template can render.

  Possible values:

  - `Template`: Every hostname `global.modules.publicDomainTemplate` can render, plus its wildcard form where the template's `%s` is the whole first label. A module never has to ask to be covered, and a hostname is reserved as soon as the template can produce it, whether or not any Deckhouse module publishes it today. A hostname that a workload legitimately serves is given back with `excludedServices` below, or with the per-namespace label. The wildcard form is the exception: it is reserved by exact match and `excludedServices` cannot free it, so a workload that serves it needs the per-namespace label or `List` below.
  - `List`: Only the hostnames of the services DP knows it publishes. The hostnames in the list are hard-specified in the `reservedNames` parameter of the Helm template, which is included with the release and maintained manually. Any public domain not included in the list—which DP will begin publishing later—remains unreserved until the list is updated in a DP release.

  `Template` applies when `%s` is the whole first label (`%s.example.com`) or a suffix of it (`kube-%s.example.com`). A template where `%s` is only a prefix of the first label (`%s-cluster.example.com`) is reserved as `List` instead: that shape can render every hostname sharing the rest of the label, including tenant Ingresses named the same way.

  The hostnames workloads already serve are recorded once, when `Template` first takes effect, and stay allowed. Switching from `List` to `Template` later takes that record then, so a workload that appeared while `List` was in force is grandfathered too.

  The record is taken once and is not taken again. Switching to `List` and back to `Template` keeps the record of the first time, so a workload that claimed a reserved hostname in between is denied on its next write. Recording again on every return would make the reservation avoidable by switching to `List`, claiming the hostname and switching back.

  Ecosystem applications are unaffected either way: `global.applications.publicDomainTemplate` puts the application name and its namespace in two different labels, so an application hostname has one label more than anything this reservation covers.

  Default: `List`

  Allowed values: `Template`, `List`

  Examples:

  ```
  mode: Template
  ```

  ```
  mode: List
  ```

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, `tolerations` will be determined automatically.

  Caution! Deckhouse will stop working if `tolerations` specified are incorrect. You need to change the values to the correct ones in `ModuleConfig/deckhouse` and `deployment/deckhouse` to get Deckhouse back on track.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string
- **settings.update**
  object

  Settings of the Deckhouse update mode and windows.

  Example:

  ```
  update:
    windows:
    - from: '8:00'
      to: '15:00'
      days:
      - Tue
      - Sat
    disruptionApprovalMode: Manual
    notification:
      webhook: https://release-webhook.mydomain.com
      minimalNotificationTime: 6h
      auth:
        basic:
          username: user
          password: password
  ```

  - **settings.update.blockOnAlerts**
    object
    - **settings.update.blockOnAlerts.enabled**
      boolean

      If enabled, the update will be blocked if there are any alerts in the cluster with severity level greater than or equal to the specified severity level.

      Default: `false`

    - **settings.update.blockOnAlerts.severity**
      integer

      The severity level of the alerts that will block the update. The severity level is a number from 0 to 9, where 0 is the highest severity and 9 is the lowest severity.

      Default: `4`

      Allowed values: `0 <= X <= 9`

  - **settings.update.disruptionApprovalMode**
    Deprecated
    string

Update mode for disruptive Deckhouse releases:

- `Auto` — a disruptive release is approved automatically.
- `Manual` — requires a manual release confirmation (set the `release.deckhouse.io/disruption-approved=true` annotation on the appropriate DeckhouseRelease resource to apply the update).

Default: `Auto`

Allowed values: `Auto`, `Manual`

- **settings.update.mode**
  string

  Update mode of Deckhouse on the selected release channel.

  - `AutoPatch` — automatic update mode for patch releases.

    To change a minor version (for example, from `v1.69.*` to `v1.70.*`), confirmation is required.

    A patch version update (for example, from `v1.70.1` to `v1.70.2`) is applied taking into account the update windows, if they are set.

  - `Auto` — automatic update mode for all versions.

    A minor version updates (for example, from `v1.69.*` to `v1.70.*`) and patch version updates (for example, from `v1.70.1` to `v1.70.2`) are applied taking into account the update windows, if they are set.

  - `Manual` — manual update mode for all versions.

    Confirmation is required for updating both minor and patch versions.

  To confirm the version update, it is necessary to set the `approved` field to `true` in the corresponding resource DeckhouseRelease.

  Default: `AutoPatch`

  Allowed values: `AutoPatch`, `Auto`, `Manual`

- **settings.update.notification**
  object

  Settings for notifications of scheduled Deckhouse updates.

  Has the effect **only** when the automatic update mode is set.

  Example:

  ```
  notification:
    webhook: https://release-webhook.mydomain.com
    minimalNotificationTime: 8h
    retryMinTime: 2s
  ```

  - **settings.update.notification.auth**
    object

    Authentication settings for the webhook.

    If the parameter is omitted, the webhook will be called without authentication.

    - **settings.update.notification.auth.basic**
      object

      Basic authentication settings for the webhook.

      If the parameter is omitted, the webhook will be called without authentication.

      - **settings.update.notification.auth.basic.password**
        string

        Required value

        The password for the webhook.

        The username and password will be sent in the `Authorization` header in the format `Basic <base64(username:password)>`.

      - **settings.update.notification.auth.basic.username**
        string

        Required value

        The username for the webhook.

        The username and password will be sent in the `Authorization` header in the format `Basic <base64(username:password)>`.

    - **settings.update.notification.auth.bearerToken**
      string

      The token for the webhook.

      The token will be sent in the `Authorization` header in the format `Bearer <token>`.

  - **settings.update.notification.minimalNotificationTime**
    string

    The minimum time that must pass before updating from the moment a new minor version appears on the release channel used.

    It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    The update mechanism ensures that Deckhouse will not be updated until a specified period of time has passed.

    When using update windows, the Deckhouse update will happen at the nearest possible update window but not before the time specified in `minimalNotificationTime` expires.

    Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

    Example:

    ```
    minimalNotificationTime: 6h
    ```

  - **settings.update.notification.releaseType**
    string

    Defines the type of version for which the notification will be sent:

    - `Minor` — only for updating the minor version.
    - `All` — for any updates, including updating a patch version.

    Default: `Minor`

    Allowed values: `All`, `Minor`

    Example:

    ```
    releaseType: All
    ```

  - **settings.update.notification.retryMinTime**

string

The minimum time between webhook retry attempts when the webhook request fails.

It is specified as a string containing the time unit in hours, minutes, or seconds: 30s, 1m, 2m30s, 1h, 2h30m.

The retry mechanism uses exponential backoff: each subsequent retry attempt waits twice as long as the previous one. For example, if `retryMinTime` is set to `2s`, the retry delays will be: 2s, 4s, 8s, 16s, 32s.

Default value is 2 seconds if not specified.

Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m|[0-9]+s)$`

Example:

```
retryMinTime: 2s
```

- **settings.update.notification.tlsSkipVerify**
  boolean

  Skip TLS certificate verification while webhook request.

  Default: `false`

- **settings.update.notification.webhook**
  string

  URL for an external webhook handler.

  The POST request will be sent to the webhook URL after a new minor version of Deckhouse appears on the release channel before it is applied to the cluster.

  > **Caution!** If you specify an invalid webhook address, Deckhouse update will be blocked.

  > Use the minimalNotificationTime parameter if necessary to set the minimum time that must pass before updating from the moment a new minor version appears on the release channel used.

  Example of the POST request payload (`Content-Type: application/json`):

  ```
  {
    "subject":"Deckhouse",
    "version": "1.36.0",
    "requirements":  {"k8s": "1.20.0"},
    "changelogLink": "https://github.com/deckhouse/deckhouse/changelog/1.36.md",
    "applyTime": "2023-01-01T14:30:00Z00:00",
    "message": "New Deckhouse Release 1.36.0 is available. Release will be applied at: Friday, 01-Jan-23 14:30:00 UTC"
  }
  ```

  Description of POST request fields:

  - `subject` — string, the update event type. Possible values: `Deckhouse`, `Module`;
  - `moduleName` — string, the name of the module. Set only if `subject: Module`.
  - `version` - string, x.y.z (semantic versioning);
  - `requirements` - object, version requirements;
  - `changelogLink` - string, a URL to the minor version changelog;
  - `applyTime` - string, date and time of the scheduled update (taking into account the configured update windows) in RFC3339 format;
  - `message` - string, a text message about the availability of the new minor version and the scheduled update time.

  Pattern: `^https?://[^\s/$.?#].[^\s]*$`

  Example:

  ```
  webhook: https://webhook.site/#!/bc8f71ac-c182-4181-9159-6ba6950afffa
  ```

- **settings.update.windows**
  array of objects

  List of update windows during the day.

  - **settings.update.windows.days**
    array of strings

    The days of the week on which the update window is applied.

    Example:

    ```
    days:
    - Mon
    - Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

      Example:

      ```
      Mon
      ```

  - **settings.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be less than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **settings.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

Should be more than the start time of the update window.

Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Example:

```
to: '18:30'
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

The module lifecycle stage: General Availability

ConversionWebhook

Short names: cwhc

Scope: Cluster
Version: `v1alpha1`

ConversionWebhook is the Schema for the conversionwebhooks API

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
- **context**
  array of objects

  Run a hook on a Kubernetes object changes.
  - **context.kubernetes**
    object
    - **context.kubernetes.allowFailure**
      boolean

      If `true`, Shell-operator skips the hook execution errors. If `false` or the parameter is not set, the hook is restarted after a 5 seconds delay in case of an error.
    - **context.kubernetes.apiVersion**
      string

      Is an optional group and version of object API. For example, it is `v1` for core objects (Pod, etc.), `rbac.authorization.k8s.io/v1beta1` for ClusterRole and `monitoring.coreos.com/v1` for prometheus-operator.
    - **context.kubernetes.fieldSelector**
      object
      - **context.kubernetes.fieldSelector.matchExpressions**
        array of objects

        Required value
        - **context.kubernetes.fieldSelector.matchExpressions.field**
          string

          Required value
        - **context.kubernetes.fieldSelector.matchExpressions.operator**
          string

          Required value
        - **context.kubernetes.fieldSelector.matchExpressions.value**
          string
    - **context.kubernetes.includeSnapshotsFrom**
      array of strings

      An array of names of kubernetes bindings in a hook. When specified, a list of monitored objects from that bindings will be added to the binding context in a snapshots field. Self-include is also possible.
    - **context.kubernetes.jqFilter**
      string

      An optional parameter that specifies event filtering using jq syntax. The hook will be triggered on the “Modified” event only if the filter result is changed after the last event.
    - **context.kubernetes.kind**
      string

      Required value

      Is the type of a monitored Kubernetes resource. This field is required.
    - **context.kubernetes.labelSelector**
      object

      A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.
      - **context.kubernetes.labelSelector.matchExpressions**
        array of objects

        matchExpressions is a list of label selector requirements. The requirements are ANDed.

        A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.
        - **context.kubernetes.labelSelector.matchExpressions.key**
          string

          Required value

          key is the label key that the selector applies to.
        - **context.kubernetes.labelSelector.matchExpressions.operator**
          string

          Required value

operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **context.kubernetes.labelSelector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **context.kubernetes.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **context.kubernetes.nameSelector**
  object
  - **context.kubernetes.nameSelector.matchNames**
    array of strings

    Required value

- **context.kubernetes.namespace**
  object

  Filters to choose namespaces.

  - **context.kubernetes.namespace.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **context.kubernetes.namespace.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
        string

        Required value

        operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.values**
        array of strings

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **context.kubernetes.namespace.labelSelector.matchLabels**
      object

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

  - **context.kubernetes.namespace.nameSelector**
    object
    - **context.kubernetes.namespace.nameSelector.matchNames**
      array of strings

      Required value

- **context.kubernetes.queue**
  string

  A name of a separate queue. It can be used to execute long-running hooks in parallel with hooks in the “main” queue.

- **context.name**
  string

  Required value

  It is used to distinguish different bindings during runtime.

- **conversions**
  array of objects

  Required value

  This binding transforms a hook into a handler for conversions defined in CustomResourceDefinition. The Shell-operator updates a CRD with `.spec.conversion`, starts HTTPS server, and runs hooks to handle ConversionReview requests.

  - **conversions.from**
    string

    Required value

    A version of a custom resource that hook can convert.

  - **conversions.handler**
    object

    Required value

    Code of the conversion handler.

    - **conversions.handler.python**
      string

      Python script handler for object.

  - **conversions.includeSnapshotsFrom**
    array of strings

    An array of names of Kubernetes bindings in a hook. When specified, a list of monitored objects from these bindings will be added to the binding context in the snapshots field.

  - **conversions.to**
    string

    Required value

    A version of a custom resource that hook can produce.

- **kind**
  string

Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **status**
  object

  Status defines the observed state of ConversionWebhook.

ValidationWebhook

Short names: vwhc

Scope: Cluster
Version: v1alpha1

ValidationWebhook is the Schema for the validationwebhooks API

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **context**
  array of objects

  Run a hook on a Kubernetes object changes.

  - **context.kubernetes**
    object
    - **context.kubernetes.allowFailure**
      boolean

      If `true`, Shell-operator skips the hook execution errors. If `false` or the parameter is not set, the hook is restarted after a 5 seconds delay in case of an error.

    - **context.kubernetes.apiVersion**
      string

      Is an optional group and version of object API. For example, it is `v1` for core objects (Pod, etc.), `rbac.authorization.k8s.io/v1beta1` for ClusterRole and `monitoring.coreos.com/v1` for prometheus-operator.

    - **context.kubernetes.fieldSelector**
      object
      - **context.kubernetes.fieldSelector.matchExpressions**
        array of objects

        Required value

        - **context.kubernetes.fieldSelector.matchExpressions.field**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.operator**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.value**
          string
    - **context.kubernetes.includeSnapshotsFrom**
      array of strings

      An array of names of kubernetes bindings in a hook. When specified, a list of monitored objects from that bindings will be added to the binding context in a snapshots field. Self-include is also possible.

    - **context.kubernetes.jqFilter**
      string

      An optional parameter that specifies event filtering using jq syntax. The hook will be triggered on the “Modified” event only if the filter result is changed after the last event.

    - **context.kubernetes.kind**
      string

      Required value

      Is the type of a monitored Kubernetes resource. This field is required.

    - **context.kubernetes.labelSelector**
      object

      A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

      - **context.kubernetes.labelSelector.matchExpressions**
        array of objects

        matchExpressions is a list of label selector requirements. The requirements are ANDed.

        A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

        - **context.kubernetes.labelSelector.matchExpressions.key**
          string

          Required value

          key is the label key that the selector applies to.

        - **context.kubernetes.labelSelector.matchExpressions.operator**
          string

          Required value

          operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

        - **context.kubernetes.labelSelector.matchExpressions.values**
          array of strings

          values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

      - **context.kubernetes.labelSelector.matchLabels**
        object

        matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

    - **context.kubernetes.nameSelector**
      object
      - **context.kubernetes.nameSelector.matchNames**
        array of strings

Required value

- **context.kubernetes.namespace**
  object

  Filters to choose namespaces.

  - **context.kubernetes.namespace.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **context.kubernetes.namespace.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
        string

        Required value

        operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.values**
        array of strings

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **context.kubernetes.namespace.labelSelector.matchLabels**
      object

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

  - **context.kubernetes.namespace.nameSelector**
    object
    - **context.kubernetes.namespace.nameSelector.matchNames**
      array of strings

      Required value

- **context.kubernetes.queue**
  string

  A name of a separate queue. It can be used to execute long-running hooks in parallel with hooks in the "main" queue.

- **context.name**
  string

  Required value

  It is used to distinguish different bindings during runtime.

- **handler**
  object

  Required value

  Code of the ValidatingWebhook handler

  - **handler.cel**
    string

    CEL rules handler for object.

  - **handler.python**
    string

    Python script handler for object.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **status**
  object

  Status defines the observed state of ValidationWebhook.

- **validationObject**
  object

  Required value

  ValidatingWebhook describes an webhook and the resources and operations it applies to.

  - **validationObject.failurePolicy**
    string

    Defines how errors from the hook are handled.

  - **validationObject.group**
    string

    A key to include snapshots from a group of schedule and Kubernetes bindings. See grouping.

  - **validationObject.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **validationObject.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **validationObject.labelSelector.matchExpressions.key**
        string

Required value

key is the label key that the selector applies to.

- **validationObject.labelSelector.matchExpressions.operator**
  string

  Required value

  operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **validationObject.labelSelector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **validationObject.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

- **validationObject.matchConditions**
  array of objects

  An optional list of match conditions for fine-grained request filtering. Available only since v1.27 of Kubernetes.

  MatchCondition represents a condition which must by fulfilled for a request to be sent to a webhook.

  - **validationObject.matchConditions.expression**
    string

    Required value

    Expression represents the expression which will be evaluated by CEL. Must evaluate to bool. CEL expressions have access to the contents of the AdmissionRequest and Authorizer, organized into CEL variables:

    'object' - The object from the incoming request. The value is null for DELETE requests. 'oldObject' - The existing object. The value is null for CREATE requests. 'request' - Attributes of the admission request(/pkg/apis/admission/types.go#AdmissionRequest). 'authorizer' - A CEL Authorizer. May be used to perform authorization checks for the principal (user or service account) of the request. See https://pkg.go.dev/k8s.io/apiserver/pkg/cel/library#Authz 'authorizer.requestResource' - A CEL ResourceCheck constructed from the 'authorizer' and configured with the request resource. Documentation on CEL: https://kubernetes.io/docs/reference/using-api/cel/

    Required.

  - **validationObject.matchConditions.name**
    string

    Required value

    Name is an identifier for this match condition, used for strategic merging of MatchConditions, as well as providing an identifier for logging purposes. A good name should be descriptive of the associated expression. Name must be a qualified name consisting of alphanumeric characters, *'-', '.' or '.', and must start and end with an alphanumeric character (e.g. 'MyName', or 'my.name', or '123-abc', regex used for validation is '([A-Za-z0-9][-A-Za-z0-9.]*)?[A-Za-z0-9]')* with an optional DNS subdomain prefix and '/' (e.g. 'example.com/MyName')

    Required.

- **validationObject.name**
  string

  Required value

  Name should be a domain with at least three segments separated by dots.

- **validationObject.namespace**
  object

  Namespace filtering. Only labelSelector is supported (not nameSelector). To filter by namespace name, use labelSelector with kubernetes.io/metadata.name label.

  - **validationObject.namespace.labelSelector**
    object

    labelSelector is a selector to filter namespaces by labels. Use kubernetes.io/metadata.name label to filter by namespace name.

    - **validationObject.namespace.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **validationObject.namespace.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

      - **validationObject.namespace.labelSelector.matchExpressions.operator**
        string

        Required value

        operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **validationObject.namespace.labelSelector.matchExpressions.values**
        array of strings

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **validationObject.namespace.labelSelector.matchLabels**
      object

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

- **validationObject.rules**
  array of objects

  A required list of rules used to determine if a request to the Kubernetes API server should be sent to the hook.

  RuleWithOperations is a tuple of Operations and Resources. It is recommended to make sure that all the tuple expansions are valid.

  - **validationObject.rules.apiGroups**
    array of strings

    APIGroups is the API groups the resources belong to. *' ' is all groups. If ''* is present, the length of the slice must be one. Required.

  - **validationObject.rules.apiVersions**
    array of strings

    APIVersions is the API versions the resources belong to. *' ' is all versions. If ''* is present, the length of the slice must be one. Required.

  - **validationObject.rules.operations**

array of strings

Operations is the operations the admission hook cares about - CREATE, UPDATE, DELETE, CONNECT or * for all of those operations and any future admission operations that are added. If '*' is present, the length of the slice must be one. Required.

- **Element of the array**
  string

  OperationType specifies an operation for a request.

- **validationObject.rules.resources**
  array of strings

  Resources is a list of resources this rule applies to.

  For example: 'pods' means pods. 'pods/log' means the log subresource of pods. ' *' means all resources, but not subresources. 'pods/*'* means all subresources of pods. *'/scale' means all scale subresources. '/*'* means all resources and their subresources.

  If wildcard is present, the validation rule will ensure resources do not overlap with each other.

  Depending on the enclosing object, subresources might not be allowed. Required.

- **validationObject.rules.scope**
  string

  scope specifies the scope of this rule. Valid values are "Cluster", "Namespaced", and " *" "Cluster" means that only cluster-scoped resources will match this rule. Namespace API objects are cluster-scoped. "Namespaced" means that only namespaced resources will match this rule. ""* means that there are no scope restrictions. Subresources match the scope of their parent resource. Default is "*".

- **validationObject.sideEffects**
  string

  Determines whether the hook is dryRun-aware.

- **validationObject.timeoutSeconds**
  integer

  Seconds API server should wait for a hook to respond before treating the call as a failure. Default is 10 (seconds).

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

The module lifecycle stage: General Availability

Setting up the update mode

You can manage DP updates in the following ways:

- Using the `settings.update` parameter of the ModuleConfig `deckhouse` resource.
- Using the `disruptions` parameters section of the NodeGroup resource.

Update windows configuration

You can configure the time windows when Deckhouse will automatically install updates in the following ways:

- For general update management, in the `update.windows` parameter of the `deckhouse` ModuleConfig resource.
- For managing disruptive updates, in the `disruptions.automatic.windows` and `disruptions.rollingUpdate.windows` parameters of the NodeGroup resource.

An example of setting up two daily update windows — from 8 a.m. to 10 a.m. and from 8 p.m. to 10 p.m. (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: EarlyAccess
    update:
      windows:
        - from: "8:00"
          to: "10:00"
        - from: "20:00"
          to: "22:00"
```

You can also set up updates on certain days, for example, on Tuesdays and Saturdays from 6 p.m. to 7:30 p.m. (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
          days:
            - Tue
            - Sat
```

Manual update confirmation

Manual confirmation of Deckhouse version updates is provided in the following cases:

- The Deckhouse update confirmation mode is enabled.

  This means that the parameter settings.update.mode in the ModuleConfig `deckhouse` is set to `Manual` (confirmation for both patch and minor versions of Deckhouse) or `AutoPatch` (confirmation for the minor version of Deckhouse). Run the following command to confirm the update (use the corresponding Deckhouse version):

```
d8 k patch DeckhouseRelease <VERSION> --type=merge -p='{"approved": true}'
```

- If automatic application of disruptive updates is disabled for a node group.

  This means that the corresponding NodeGroup has the parameter `spec.disruptions.approvalMode` set to `Manual`.

  For updating **each** node in such a group, the node must have `update.node.deckhouse.io/disruption-approved=` annotation. Example:

  ```
  d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
  ```

Deckhouse update notifications

In the `Auto` update mode, you can configure a webhook call to receive a notification about an upcoming minor Deckhouse version update.

In addition, notifications are generated not only for Deckhouse updates but also for updates of any modules, including their individual updates. In some cases, the system may initiate the sending of multiple notifications at once (10–20 notifications) at approximately 15-second intervals.

> **i** Notifications are available only in the `Auto` update mode; in the `Manual` mode they are not generated.

> **i** Specifying a webhook is optional: if the `update.notification.webhook` parameter is not set but the `update.notification.minimalNotificationTime` parameter is specified, the update will still be postponed for the specified period. In this case, the appearance of a DeckhouseRelease resource in the cluster with the name of the new version can be considered the notification of its availability.

Notifications are sent only once for a specific update. If something goes wrong (for example, the webhook receives incorrect data), they will not be resent automatically. To resend the notification, you must delete the corresponding DeckhouseRelease resource.

Example of notification configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
```

After a new minor Deckhouse version appears on the selected release channel, but before it is applied in the cluster, a POST request will be sent to the configured webhook address.

The minimalNotificationTime parameter allows you to postpone the update installation for the specified period, providing time to react to the notification while respecting update windows. If the webhook is unavailable, each failed attempt to send the notification will postpone the update by the same duration, which may lead to the update being deferred indefinitely.

> **⚠** If your webhook returns any status code out of 2xx range, DP retries sending the notification up to five times with exponential backoff. If all attempts fail, the release is blocked until the webhook becomes available again.

For easier error handling and debugging, when returning error codes the webhook should return a JSON response with the following structure:

- `code` — optional internal error code for programmatic handling;
- `message` — a human-readable description of what went wrong.

If the webhook returns a successful HTTP status (2xx), DP treats the notification as successful regardless of the response body.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h
```

Minimal Go webhook example...

```
package main

import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
)

// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
  Subject      string          `json:"subject"`
  Version      string          `json:"version"`
  Requirements  map[string]string `json:"requirements,omitempty"`
  ChangelogLink string          `json:"changelogLink,omitempty"`
  ApplyTime    string          `json:"applyTime,omitempty"`
  Message      string          `json:"message"`
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
  Code    string `json:"code,omitempty"`
  Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
  if r.Method != http.MethodPost {
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  defer r.Body.Close()

  var data WebhookData
  if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
    log.Printf("failed to decode payload: %v", err)
    w.WriteHeader(http.StatusInternalServerError)
    return
  }

  // Print payload fields
  log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
    data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
  log.Printf("message=%s", data.Message)

  // Example conditional logic: fail intentionally for testing
  if data.Version == "v0.0.0-fail" {
    // Return structured error response with error status code
    errorResp := ResponseError{
      Code:    "TEST_FAILURE",
      Message: "intentional failure for testing",
    }

    w.WriteHeader(http.StatusBadRequest)
    json.NewEncoder(w).Encode(errorResp)
    return
  }

  // Return success response with 2xx status code
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Notification processed successfully"))
}

func main() {
  mux := http.NewServeMux()
  mux.HandleFunc("/webhook", handler)

  addr := ":8080"
  fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
  if err := http.ListenAndServe(addr, mux); err != nil {
    log.Fatal(err)
  }
}
```

Reserving the hostnames of the platform web interfaces

DP publishes its own web interfaces under hostnames rendered from the `publicDomainTemplate` global parameter: the API, the web interface, Grafana, Dex, the kubeconfig generator and others. Those hostnames are reserved cluster-wide. An Ingress, HTTPRoute, GRPCRoute, TLSRoute, ListenerSet or Gateway resource created outside a namespace labeled `heritage: deckhouse` cannot claim such a name, and such a request is rejected with a message naming the hostname:

```
Hostname console.example.com is reserved for Deckhouse platform services
and cannot be claimed outside a system namespace
```

Only resource creation and modification are checked. An object that already claims a reserved hostname keeps serving traffic until it is modified.

Reserved hostnames

The reserved set is controlled by the `settings.reservedPublicHosts.mode` parameter of the ModuleConfig `deckhouse`:

- `Template` (by default): Every hostname the template can render are reserved.
- `List`: Only the hostnames of the services DP knows it publishes.

For details on each reservation method, refer to the parameter description.

Excluding a name from reservation

Under `Template` mode, the reservation covers not only the hostnames used by DP but also others matching the template. A workload that requires such a hostname can be excluded from reservation in one of the following ways:

- Free a single hostname with the `settings.reservedPublicHosts.excludedServices` parameter.

  The parameter includes not the entire hostname, only the service name (the value included in the template). For example, for the `%s.example.com` template, the `grafana` value excludes `grafana.example.com` from reservation.
- Exclude a whole namespace with the `security.deckhouse.io/reserved-hosts-bypass: "true"` label.

The label is set on the namespace, not on the object, so a tenant who is allowed to manage resources only in their own namespaces can't set it.

- Return to the previous reservation by setting `mode: List`.

The following is an example that frees `grafana.example.com` from reservation and additionally reserves `admin.corp.example.org` that doesn't match the template:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    reservedPublicHosts:
      mode: Template
      excludedServices:
        - grafana
      additionalHosts:
        - admin.corp.example.org
```

The `settings.reservedPublicHosts.additionalHosts` parameter lets you reserve additional hostnames regardless of the `excludedServices` value.

⚠ The `excludedServices` parameter accepts service names only, therefore it can't be used to exclude a wildcard name from reservation.

To allow a workload to use a wildcard name (such as `*.example.com`), disable the checking for its namespace using the above-mentioned label or use the `List` mode.

Excluding the shared Gateway

The Gateway to which DP module ListenerSet resources are attached is automatically excluded from reservation, regardless of the hostnames specified in its listeners.

The following Gateway is considered the shared Gateway:

- The Gateway specified in the global `gatewayAPIGateway` parameter.
- If the parameter is not set, the Gateway specified in the `default-gateway` ConfigMap in the `d8-alb` namespace.

The exclusion applies by name and namespace and only to the Gateway resource. It does not apply to other resources using the same hostnames, such as Ingress, HTTPRoute, or another Gateway.

This exclusion is necessary because such a Gateway typically has a listener with a wildcard hostname and wildcard certificate and is often located in a dedicated namespace without the `heritage: deckhouse` label. Without the exclusion, the Gateway could not be modified after hostname reservation is enabled, even though the attached DP routes continue to depend on it.

Any other Gateway with a `*.example.com` listener is rejected, even if that hostname was already in use when reservation was enabled. To use such a Gateway, add the `security.deckhouse.io/reserved-hosts-bypass: "true"` label to its namespace or enable `List` mode.

If the shared Gateway is neither specified in the `gatewayAPIGateway` parameter nor defined through the `default-gateway` ConfigMap, no automatic exclusion is applied.

Reservation with existing applications in the cluster

When hostname reservation takes effect, hostnames already used by applications are automatically added to the list of allowed hostnames. Otherwise, the next modification of the corresponding resources would be rejected.

The list is stored in the `grandfatheredHosts` key of the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace, separately from hostnames manually excluded from reservation.

The list is created only when `Template` mode is enabled for the first time and is not updated afterward. This prevents reservation from being bypassed: an application cannot claim a reserved hostname after reservation has been enabled and then have that hostname allowed through subsequent regeneration of the list.

If an application that used a hostname allowed this way is removed and the hostname is no longer needed, remove the corresponding entry from `grandfatheredHosts`.

Before being stored, hostnames are converted to lowercase and any trailing dot is removed. Invalid values, such as a URL instead of a hostname, are ignored. If a resource uses the corresponding reserved hostname, its next modification will be rejected.

Wildcard hostnames, such as `*.example.com`, are not added to `grandfatheredHosts`. Otherwise, an application could use any hostname matching the template, including hostnames of services that DP may publish in the future. To allow an application to use a wildcard hostname in the platform domain, add the `security.deckhouse.io/reserved-hosts-bypass: "true"` label to its namespace or enable `List` mode.

This restriction does not apply to the Gateway to which DP is attached. This Gateway is automatically excluded from the check.

Switching from `Template` mode to `List` mode and back does not regenerate `grandfatheredHosts`. Therefore, hostnames that applications start using while `List` mode is enabled are not added to the list of allowed hostnames after switching back to `Template` mode. When the corresponding resources are modified, these hostnames are checked according to the standard reservation rules.

Checking reservation status

Information about the current reservation status is stored in the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace.

To view the ConfigMap, run the following command:

```
d8 k -n d8-system get configmap d8-reserved-public-hosts -o yaml
```

The ConfigMap contains the following keys:

Key	Description
`mode`	Current reservation mode. If `publicDomainTemplate` does not match the schema, the value is `List`
`hostPattern`	Regular expression describing the hostnames that can be generated from the template. The value is empty in `List` mode
`hosts`	Hostnames matched exactly. In `Template` mode, contains the values from `additionalHosts` and the wildcard domain name; in `List` mode, contains the hostnames of platform services
`allowedHosts`	Hostnames excluded from matching against `hostPattern`: hostnames from `excludedServices` and `grandfatheredHosts`
`excludedHosts`	Hostnames excluded from reservation using `excludedServices`
`grandfatheredHosts`	Hostnames that were already used by applications when reservation was enabled and were automatically added to the list of allowed hostnames
`sharedGateway`	Gateway to which DP is attached, in the `<NAMESPACE>/<NAME>` format. The value is empty if no such Gateway is configured or detected
`platformHosts`	Hostnames of services published by DP. Used as a reference in both reservation modes

Key	Description
`unknownExcludedServices`	Names from `excludedServices` that are not published by any module. A value in this key most likely indicates a typo in the service name

Collect debug info

Read the FAQ to learn more about collecting debug information.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How to run kube-bench in my cluster?

First, you have to exec in Deckhouse Pod:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- bash
```

Then you have to select which node you want to run kube-bench.

- Run on random node:

  ```
  curl -s https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k create -f -
  ```

- Run on specific node, e.g. control-plane node:

  ```
  curl -s https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k apply -f - --dry-run=client -o json | jq
  '.spec.template.spec.tolerations=[{"operator": "Exists"}] | .spec.template.spec.nodeSelector={"node-role.kubernetes.io/control-
  plane": ""}' | d8 k create -f -
  ```

Then you can check report:

```
d8 k logs job.batch/kube-bench
```

> ⚠ Deckhouse set the log retention period to 7 days. However, according to the security requirements specified in kube-bench, logs should be retained for at least 30 days. Use separate storage for logs if you need to keep logs for more than 7 days.

How to collect debug info?

1. Create a diagnostic archive with the `d8` utility, redirecting its output (stdout) to a file:

   ```
   d8 system collect-debug-info > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz
   ```

2. Send the resulting archive to the Deckhouse team for further debugging.

> The `--exclude` flag omits the specified items from the archive. Example:

```
d8 system collect-debug-info --exclude=queue global-values > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz
```

> The `--list-exclude` flag prints the list of items available for exclusion. Example:

```
d8 system collect-debug-info --list-exclude
```

The following information is produced when creating the archive. Names in the “File in archive” column correspond to top-level items inside the resulting `tar.gz` archive. Certain sensitive values (e.g., `kubeRBACProxyCA` and `registry.dockercfg`) are excluded.

Category	Collected data	File in archive
Deckhouse	Deckhouse queue state	`queue`
	Deckhouse values (excluding `kubeRBACProxyCA` and `registry.dockercfg`)	`global-values`
	Current `deckhouse` pod version	`deckhouse-version`
	All `DeckhouseRelease` objects	`deckhouse-releases`
	Deckhouse pod logs	`deckhouse-logs`
	Manifests of controllers and pods from all Deckhouse namespaces	`d8-all`
Cluster objects	`NodeGroup`	`node-groups`
	`NodeGroupConfiguration`	`node-group-configuration`
	`Node`	`nodes`
	`Machine`	`machines`
	`Instance`	`instances`
	`StaticInstance`	`staticinstances`
	`MachineDeployment`	`cloud-machine-deployment`, `static-machine-deployment`
	`ClusterAuthorizationRule`	`cluster-authorization-rules`
	`AuthorizationRule`	`authorization-rules`
	`ModuleConfig`	`module-configs`
	Events (all namespaces)	`events`

Category	Collected data	File in archive
Modules and states	List of enabled modules	`deckhouse-enabled-modules`
	`ModuleSource` objects in the cluster	`deckhouse-module-sources`
	`ModulePullOverride` objects in the cluster	`deckhouse-module-pull-overrides`
	Modules in `maintenance` mode	`deckhouse-maintenance-modules`
Controller logs and manifests	`machine-controller-manager` logs	`mcm-logs`
	`cloud-controller-manager` logs	`ccm-logs`
	`csi-controller` logs	`csi-controller-logs`
	`cluster-autoscaler` logs	`cluster-autoscaler-logs`
	Vertical Pod Autoscaler admission controller logs	`vpa-admission-controller-logs`
	Vertical Pod Autoscaler recommender logs	`vpa-recommender-logs`
	Vertical Pod Autoscaler updater logs	`vpa-updater-logs`
	`capi-controller-manager` YAML	`capi-controller-manager`
	`caps-controller-manager` YAML	`caps-controller-manager`
	`machine-controller-manager` YAML	`machine-controller-manager`
Monitoring and alerts	Prometheus logs	`prometheus-logs`
	Active (firing) Prometheus alerts	`alerts`
	Pods not in `Running` (excluding `Completed` and `Evicted`)	`bad-pods`
	List of Audit Policies	`audit-policy`
Network	All objects in the `d8-istio` namespace	`d8-istio-resources`
	All `istio` custom resources	`d8-istio-custom-resources`
	Envoy configuration for `istio`	`d8-istio-envoy-config`
	`istio` logs	`d8-istio-system-logs`
	`istio` ingress gateway logs	`d8-istio-ingress-logs`
	`istio` users logs	`d8-istio-users-logs`
	Cilium connection status (`cilium health status`)	`cilium-health-status`

How to debug pod problems with ephemeral containers?

Run the following command:

```
d8 k -n <namespace_name> debug -it <pod_name> --image=ubuntu <container_name>
```

More info in official documentation.

An ephemeral container cannot be added to a Pod labeled `heritage: deckhouse`, see the next section.

Who can run exec into Deckhouse Platform pods and scale Deckhouse Platform objects?

Only Deckhouse Platform system users. For all other users, including those with the `ClusterAdmin` and `SuperAdmin` roles, the `deckhouse` module prohibits the following operations in any namespace:

- `kubectl exec`, `kubectl attach`, `kubectl cp`, `kubectl port-forward` and `kubectl debug` into a Pod labeled `heritage: deckhouse`. The restriction also applies to `kubectl port-forward` to Services if the request is directed to a protected Pod.
- `kubectl scale` of Deployment, StatefulSet, ReplicaSet or ReplicationController resources labeled `heritage: deckhouse`, unless the object is under maintenance, that is, labeled `maintenance.deckhouse.io/no-resource-reconciliation`.

The `heritage: deckhouse` label is checked on the object itself. For example, if a Deployment has this label but a Pod created by it does not, the access restrictions do not apply to that Pod.

These restrictions do not apply to the following users and components:

- `system:sudouser`.
- `dhctl` and `observability`.
- Service accounts from `d8-*` and `kube-*` namespaces (`system:serviceaccount:d8-*`, `system:serviceaccount:kube-*`), including the Deckhouse Platform controllers and the Horizontal Pod Autoscaler from `kube-controller-manager`.
- Kubernetes control plane components (`system:apiserver`, `system:kube-controller-manager`, `system:kube-scheduler`, `system:volume-scheduler`) and nodes belonging to the `system:nodes` group.
- Members of the `system:masters` group, that is, when using `super-admin.conf` on a master node. The restrictions still apply when using `admin.conf` (the `kubeadm:cluster-admins` group).

Except for `system:masters`, these are the same users that are allowed to modify resources with the `heritage: deckhouse` label. The restrictions apply regardless of whether the `admission-policy-engine` module is enabled in the cluster.

Requests are validated by the `webhook-handler` Pods in the `d8-system` namespace. If these Pods are unavailable, requests are rejected. In this case, the following operations are blocked in all namespaces for all users except those listed above, regardless of whether the `heritage: deckhouse` label is present:

- `kubectl exec`, `kubectl attach`, `kubectl cp`, and `kubectl port-forward` for any Pod.
- `kubectl scale` for any Deployment, StatefulSet, ReplicaSet, or ReplicationController resource.

To access a Pod for diagnostics, you can use `system:sudouser` or `super-admin.conf` on a master node.

Example of executing a command in a Pod as `system:sudouser`:

```
d8 k --as=system:sudouser -n d8-system exec -it <POD_NAME> -- sh
```

⚠ You must have permission to impersonate the `system:sudouser` user.

How to debug node problems with ephemeral containers?

Run the following command:

```
d8 k debug node/mynode -it --image=ubuntu
```

More info in official documentation.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The deckhouse-tools module

> The module lifecycle stage: General Availability

> Web interfaces associated with the module: deckhouse-tools

The module creates a web UI with links to download Deckhouse CLI tool for various operating systems.

The web interface address is composed based on the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

For example, if `publicDomainTemplate` is set to `%s-kube.company.my`, the tools web interface will be exposed at `tools-kube.company.my`.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

An example of the configuration

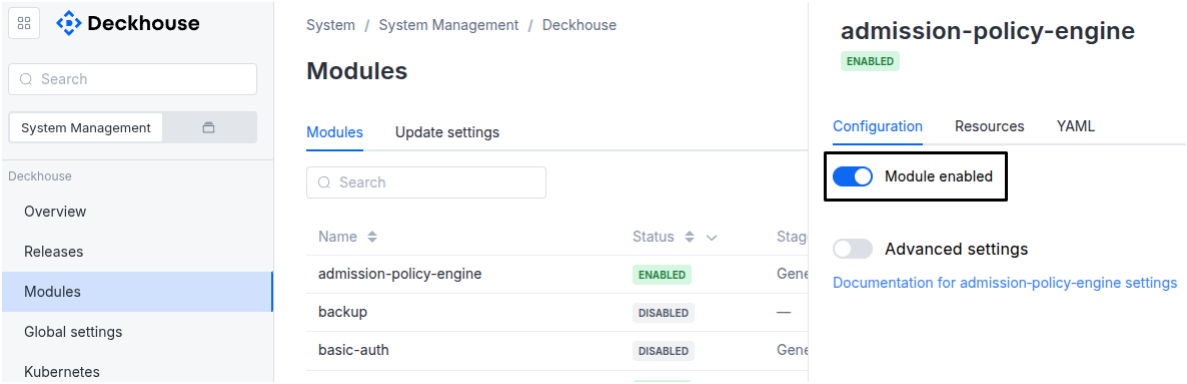
```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  enabled: true
  version: 1
```

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse-tools` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable deckhouse-tools
  ```

- **Using ModuleConfig `deckhouse-tools`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `deckhouse-tools` (create it if necessary);

  Example of a manifest to enable module `deckhouse-tools`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse-tools
  spec:
    enabled: true
  ```

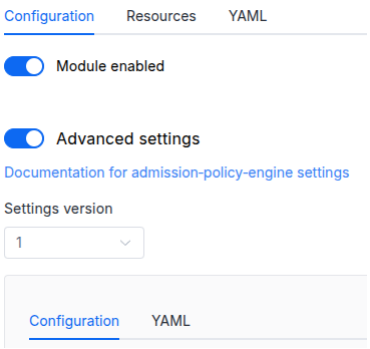
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse-tools` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `deckhouse-tools` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `deckhouse-tools`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `deckhouse-tools` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc deckhouse-tools
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `deckhouse-tools` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `deckhouse-tools`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse-tools
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.auth**
    object

    Parameters to authenticate and authorize access to the module’s web interface.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of groups whose users can browse the module’s web interface.

      This parameter is used if the `user-authn` module is enabled or if the `externalAuthentication` parameter is set.

      > **Caution!** Note that you must add those groups to the appropriate field in the `DexProvider` config if this module is used together with the `user-authn` one.

      Default: `[]`

      Examples:

      ```
      allowedUserGroups:
      - admin
      - users
      ```

      ```
      allowedUserGroups: []
      ```

    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

      > External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response).

        Default:

        Example:

        ```
        authSignInURL: https://$host/dex-authenticator/sign_in
        ```

      - **settings.auth.externalAuthentication.authURL**
        string

The URL of the authentication service.

If the user is authenticated, the service should return an HTTP 200 response code.

Default:

Example:

```
authURL: https://deckhouse-tools-dex-authenticator.d8-system.svc.cluster.local/dex-authenticator/auth
```

- **settings.https**
  object

  What certificate type to use.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: Disabled
  ```

  ```
  https:
    mode: OnlyInURI
  ```

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object

    Parameters for certmanager.

    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are supported; you can also define your own).

      Default: `letsencrypt`

      Example:

      ```
      clusterIssuerName: letsencrypt
      ```

  - **settings.https.customCertificate**
    object

    Parameters for custom certificate usage.

    - **settings.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with the module's web interface.

      Note that this secret must conform to the kubernetes.io/tls format.

  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace.
    - `Disabled` — in this mode, the module's web interface can only be accessed over HTTP.
    - `OnlyInURI` — the module's web interface will work over HTTP (implies there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. The load balancer should provide redirection from HTTP to HTTPS.

    Default: `CertManager`

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller of the module web interface.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The descheduler module

The module lifecycle stage: General Availability

Every 15 minutes, the module analyzes the cluster state and performs pod eviction according to the conditions described in the active strategies. Evicted pods go through the scheduling process again, considering the current state of the cluster. This helps redistribute workloads according to the chosen strategy.

The module is based on the descheduler project.

Features of the module

- The module can take into account the pod priority class (parameter spec.priorityClassThreshold), restricting its operation to only those pods that have a priority class lower than the specified threshold;
- The module does not evict pods in the following cases:
  - a pod is in the `d8-*` or `kube-system` namespaces;
  - a pod has a `priorityClassName` `system-cluster-critical` or `system-node-critical`;
  - a pod is associated with a local storage;
  - a pod is associated with a DaemonSet;
  - pod eviction will violate Pod Disruption Budget (PDB);
  - there are no available nodes to run the evicted pod.
- Pods with the `Best effort` priority class are evicted before those with `Burstable` and `Guaranteed`.

Descheduler uses parameters with the `labelSelector` syntax from Kubernetes to filter pods and nodes:

- `podLabelSelector` — limits pods by labels;
- `namespaceLabelSelector` — filters pods by namespaces;
- `nodeLabelSelector` — selects nodes by labels.

Metrics provider

If the cluster has a registered `metrics.k8s.io` API group (for example, when metrics-server is installed), the module automatically detects it and enables the **KubernetesMetrics** metrics provider in the `descheduler` policy. This allows to use the **actual** resource consumption data from the Metrics API instead of relying solely on pod requests and limits when the `LowNodeUtilization` strategy is set. Accordingly, the values in thresholds and targetThresholds are then interpreted as a percentage of actual consumption, not of pod requests. The `HighNodeUtilization` strategy is not affected: it always uses pod requests and limits.

No user action is required: the module discovers the `metrics.k8s.io` API group by watching APIService resources and configures the policy accordingly. If metrics-server is installed **after** the module is already running, the `descheduler` Pod will be automatically restarted with the updated policy.

If the `metrics.k8s.io` API group is not registered in the cluster, the module estimates resource usage from pod requests and limits.

Evictions in background

The module runs the descheduler with the upstream `EvictionsInBackground` feature gate enabled (`--feature-gates EvictionsInBackground=true`). There is no module setting for it: the behavior is part of the module and requires no configuration.

By default, the descheduler expects every eviction to complete immediately: the eviction API request either succeeds or is rejected. Some workloads cannot be evicted that way — for example, a pod backing a virtual machine has to be live-migrated first, which may take minutes and may need retries.

With the feature gate enabled, an eviction of a pod annotated with `descheduler.alpha.kubernetes.io/request-evict-only` is treated as a *request* to start the eviction. The controller that owns the pod rejects the eviction API call with `429 Too Many Requests` and marks the pod with the `descheduler.alpha.kubernetes.io/eviction-in-progress` annotation while it performs the actual eviction (for example, a live migration). The descheduler keeps track of such pods and counts them against its eviction limits until the pod is gone or the annotation is removed, so it doesn't start more evictions than allowed while the previous ones are still in progress.

Pods without the `descheduler.alpha.kubernetes.io/request-evict-only` annotation are evicted as usual, so nothing changes for clusters where no workload implements its own eviction policy.

The feature gate is in the alpha stage in the upstream descheduler project (see KEP-1397). It changes nothing on its own: without a controller that answers the eviction request as described above, the descheduler behaves exactly as it did before.

Strategies

HighNodeUtilization

More compactly places pods. Requires scheduler configuration and enabling auto-scaling.

To use `HighNodeUtilization`, you must explicitly specify the high-node-utilization scheduler profile for each pod (this profile cannot be set as the default).

This strategy identifies *under utilized nodes* and evicts pods from them to redistribute them more compactly across fewer nodes.

Under utilized node — A node whose resource usage is below all the threshold values specified in the strategies.highNodeUtilization.thresholds section.

The strategy is enabled by the parameter spec.strategies.highNodeUtilization.enabled.

⚠ In GKE, you cannot configure the default scheduler, but you can use the `optimize-utilization` strategy or deploy a second custom scheduler.

⚠ Node resource usage takes into account extended resources and is calculated based on pod requests and limits (requests and limits), not actual consumption. This approach ensures consistency with the kube-scheduler, which uses a similar principle when scheduling pods on nodes.

This strategy always uses pod requests and limits, regardless of whether the `metrics.k8s.io` API is available in the cluster (see Metrics provider).

LowNodeUtilization

ℹ Loads the nodes more evenly.

This strategy identifies *under utilized nodes* and evicts pods from other *over utilized nodes*. The strategy assumes that the evicted pods will be recreated on the under utilized nodes (following normal scheduler behavior).

Under utilized node — A node whose resource usage is below all the threshold values specified in the strategies.lowNodeUtilization.thresholds section.

Over utilized node — A node whose resource usage exceeds at least one of the threshold values specified in the strategies.lowNodeUtilization.targetThresholds section.

Nodes with resource usage in the range between `thresholds` and `targetThresholds` are considered optimally utilized. Pods on these nodes will not be evicted.

The strategy is enabled by the parameter spec.strategies.lowNodeUtilization.enabled.

By default, `thresholds` and `targetThresholds` are absolute percentages. If spec.strategies.lowNodeUtilization.useDeviationThresholds is `true`, they are counted from the average utilization of all Ready nodes of the cluster: a node is under utilized if its usage is at or below the average minus `thresholds`, and over utilized if its usage is above the average plus `targetThresholds`. For example, with an average CPU utilization of 40%, `thresholds: {cpu: 5}` and `targetThresholds: {cpu: 5}` make nodes at or below 35% under utilized and nodes above 45% over utilized, so the strategy keeps the nodes within the corridor of ±5% around the average.

To evict pods gradually, set spec.strategies.lowNodeUtilization.evictionLimits.node: the maximum number of pods the strategy evicts from one node in one descheduling cycle. A virtual machine that is still migrating from the node takes a place in the limit in every cycle, so for virtual machines the parameter effectively limits how many of them migrate from one node at the same time.

⚠ If the `metrics.k8s.io` API group is **not** registered in the cluster, node resource usage takes into account extended resources and is calculated based on pod requests and limits (requests and limits), not actual consumption. This approach ensures consistency with the kube-scheduler, which uses a similar principle when scheduling pods on nodes.

If the `metrics.k8s.io` API group **is** registered (for details, see “Metrics provider”), the strategy uses the **actual** resource consumption from the Metrics API *instead of* requests and limits. In this case the values used by the descheduler are close to what `kubectl top` shows, but they no longer match the kube-scheduler’s view: a node whose requests are exhausted may still be considered under utilized, and its pods will not be evicted.

CPU utilization averaged over 15 minutes

The momentary values from the Metrics API or pod requests react to short load spikes. To balance nodes by their sustained load, set spec.strategies.lowNodeUtilization.metricsSource to `Prometheus`. The strategy then uses the node CPU utilization averaged over the last 15 minutes, which the cluster Prometheus calculates from the node-exporter data.

How the strategy works in this mode:

1. Only the `cpu` key of `thresholds` and `targetThresholds` is used, as a percentage of the node CPU. The key is required in both sections, other keys are ignored.
2. The `prometheus` module must be enabled. If it is disabled, the strategy works as with `Default`, and the Deckhouse log shows a warning.
3. With `useDeviationThresholds: true` the strategy keeps the nodes within a corridor around the 15-minute average CPU utilization of the cluster.
4. The strategy cannot estimate how much one pod adds to the average, so it evicts one pod from each over utilized node in a cycle. With `evictionLimits.node` set, it evicts up to that number of pods from each over utilized node: in this mode the parameter raises the number of evictions instead of limiting it. Do not set `evictionLimits.node` with this source.
5. The average reflects the evictions with a delay of up to 15 minutes. With the `Frequent` descheduling interval a node stays over utilized for several cycles after the evictions and loses pods again. Use the `Moderate` (15 minutes, default) or `Rare` (30 minutes) interval with this source.
6. If Prometheus is unavailable or has no data for at least one Ready node (node-exporter does not run on it or fails to be scraped, or the node has just joined and has not been scraped twice yet), the strategy of this Descheduler skips the cycle. Other Descheduler resources keep working.

The descheduler reads the metrics through `aggregating-proxy` in the `d8-monitoring` namespace with its ServiceAccount token. The module grants the read access only while at least one Descheduler uses this mode. An example is on the “Examples” page.

Eviction target check

Since Deckhouse 1.78, in clusters with the `control-plane-manager` module enabled, the strategy requests the expected target node from kube-scheduler before evicting a pod. kube-scheduler runs a dry-run scheduling cycle for the pod, with the source node excluded, and returns the nodes with the top score.

How the strategy uses the answer:

1. The pod is evicted only if every node with the top score is under utilized in this cycle by this Descheduler’s own `thresholds`. With `useDeviationThresholds: true` this means at or below the average minus `thresholds`. The check has no thresholds of its own.
2. The pod stays on the node, and the strategy tries the next pod of the same node, if kube-scheduler would place the pod on a node that is not under utilized or has no node for it.
3. The pod also stays if kube-scheduler does not answer or answers with an error: the strategy does not evict a pod without the check.
4. kube-scheduler pauses the real scheduling while a check runs, so the checks are limited. After 5 checks of one over utilized node that did not allow an eviction, the node waits for the next cycle. After 100 checks in a cycle, this Descheduler resource stops evicting pods until the next cycle. If the check finds no feasible node or any top-scoring node is not underutilized, other pods of the same controller on the source node are skipped for this cycle.

The check applies to every Descheduler with `lowNodeUtilization` and cannot be turned off. On managed Kubernetes (the `control-plane-manager` module is disabled) the strategy works without it. The `descheduler_simulate_target_check_total` metric counts the checks by result.

RemoveDuplicates

ℹ Prevents multiple pods from the same controller (ReplicaSet, ReplicationController, StatefulSet) or the same Job from running on the same node.

The strategy ensures that no more than one pod of a ReplicaSet, ReplicationController, StatefulSet, or pods of a single Job is running on the same node. If there are two or more such pods, the module evicts the excess pods so that they are better distributed across the cluster.

The situation can occur if some nodes in the cluster have failed for any reason, and the pods from those nodes have been moved to other nodes. Once the failed nodes become available again to accept load, this strategy can be used to evict duplicate pods from other nodes.

The strategy is enabled by the parameter strategies.removeDuplicates.enabled.

RemovePodsHavingTooManyRestarts

The strategy evicts pods having too many restarts from nodes. Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold. This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

RemovePodsViolatingInterPodAntiAffinity

ℹ Evicts pods violating inter-pod affinity and anti-affinity rules to ensure compliance.

The strategy ensures that pods violating inter-pod affinity and anti-affinity rules are evicted from nodes.

For example, if there is **podA** on a node and **podB** and **podC** (running on the same node) have anti-affinity rules which prohibit them to run on the same node, then **podA** will be evicted from the node so that **podB** and **podC** could run. This issue could happen, when the anti-affinity rules for **podB** and **podC** are created when they

are already running on node.

The strategy is enabled by the parameter spec.strategies.highNodeUtilization.enabled.

RemovePodsViolatingNodeAffinity

> i Evicts pods violating node affinity rules to ensure compliance.

The strategy makes sure all pods violating node affinity are eventually removed from nodes.

Essentially, depending on the settings of the parameter strategies.removePodsViolatingNodeAffinity.nodeAffinityType, the strategy temporarily implement the rule `requiredDuringSchedulingIgnoredDuringExecution` of the pod's node affinity as the rule `requiredDuringSchedulingRequiredDuringExecution`, and the rule `preferredDuringSchedulingIgnoredDuringExecution` as the rule `preferredDuringSchedulingPreferredDuringExecution`.

Example for `nodeAffinityType: requiredDuringSchedulingIgnoredDuringExecution`. There is a pod scheduled to a node which satisfies the node affinity rule `requiredDuringSchedulingIgnoredDuringExecution` at the time of scheduling. If over time this node no longer satisfies the node affinity rule, and there is another node available that satisfies the node affinity rule, the strategy evicts the pod from the node it was originally scheduled to.

Example for `nodeAffinityType: preferredDuringSchedulingIgnoredDuringExecution`. There is a pod scheduled to a node because at the time of scheduling there were no other nodes that satisfied the node affinity rule `preferredDuringSchedulingIgnoredDuringExecution`. If over time an available node that satisfies this rule appears in the cluster, the strategy evicts the pod from the node it was originally scheduled to.

The strategy is enabled by the parameter strategies.removePodsViolatingNodeAffinity.enabled.

RemovePodsViolatingTopologySpreadConstraint

The strategy ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint's `maxSkew`. This is useful for rebalancing pods across zones after a zone outage recovery.

Third-party components

List of third-party software used in the `descheduler` module:

- Descheduler 0.37.0

  License: Apache License 2.0

  A tool to auto-fix scheduling issues Kubernetes does not cover itself.

> i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

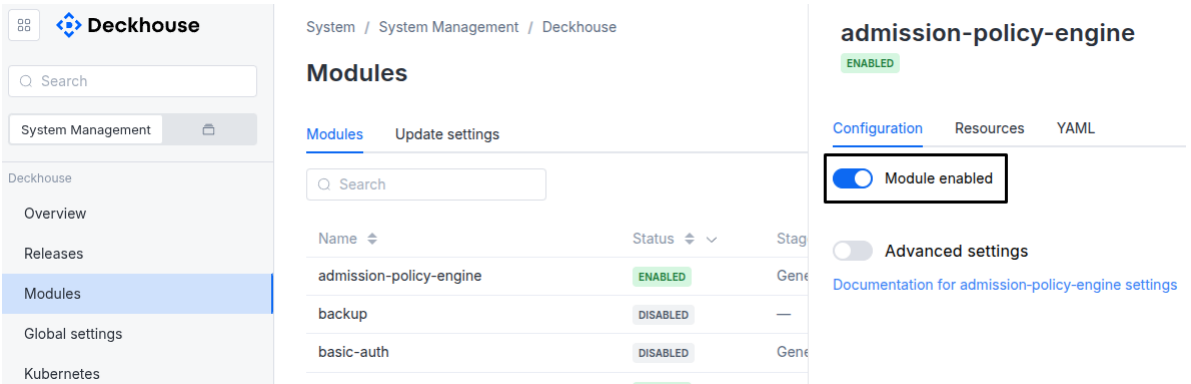
> i The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `descheduler` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable descheduler
```

- **Using ModuleConfig `descheduler`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `descheduler` (create it if necessary);

  Example of a manifest to enable module `descheduler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true
```

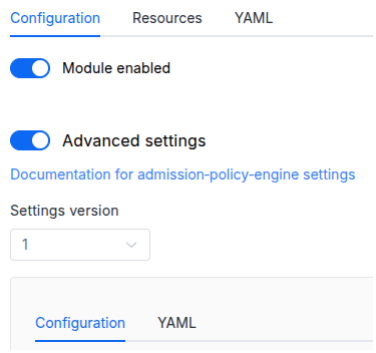
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `descheduler` module and enable the "Advanced Settings" switch. Fill in the required fields in the "Configuration" tab or specify the module settings in YAML format on the "YAML" tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `descheduler` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `descheduler`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `descheduler` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc descheduler
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `descheduler` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `descheduler`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: descheduler
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `settings`.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.deschedulingInterval**
    string

    Interval between pod redistribution runs.

    Supported values:

    - `Frequent`: Runs every 5 minutes. Suitable for clusters where faster pod redistribution is important.
    - `Moderate`: Runs every 15 minutes (default).
    - `Rare`: Runs every 30 minutes. Suitable for clusters where it’s important to minimize the number of pod redistributions.

    Default: `Moderate`

    Allowed values: `Frequent`, `Moderate`, `Rare`
  - **settings.logLevel**
    string

    Descheduler logging level.

    Each level includes the messages of the less detailed levels. Some informational messages and warnings, such as startup messages, are logged at any level.

    Supported values:

    - `Error`: Errors only.
    - `Warning`: Configuration warnings, evicted pods, and descheduling cycle results (default).
    - `Info`: The progress of each descheduling cycle, such as the nodes being processed.
    - `Debug`: Detailed information on eviction decisions, such as why a pod was not evicted, and every request to the Kubernetes API. Produces a very large volume of logs, so use it for troubleshooting only.

    Default: `Warning`

    Allowed values: `Debug`, `Info`, `Warning`, `Error`

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> **i** The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

i The module lifecycle stage: General Availability

Descheduler

Scope: Cluster

- v1alpha2
- v1alpha1

Descheduler is a description of a single descheduler instance.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
  - **metadata.name**
    string
- **spec**
  object

  Required value

  Defines the behavior of a descheduler instance.

  - **spec.evictLocalStoragePods**
    boolean

    Allows Pods using local storage to be evicted.

    Default: `false`

  - **spec.namespaceLabelSelector**
    object

    Limiting the pods which are processed by namespace labels.

    - **spec.namespaceLabelSelector.matchExpressions**
      array of objects

      List of label expressions that a node should have to qualify for the filter condition.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      - key: tier
        operator: NotIn
        values:
        - production
      ```

      - **spec.namespaceLabelSelector.matchExpressions.key**
        string

        A label name.

      - **spec.namespaceLabelSelector.matchExpressions.operator**
        string

        A comparison operator.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.namespaceLabelSelector.matchExpressions.values**
        array of strings

        A label value.

        - **Element of the array**
          string

          Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

          Length: `1..63`

    - **spec.namespaceLabelSelector.matchLabels**
      object
  - **spec.nodeLabelSelector**
    object

    Limiting the pods which are processed to fit evicted pods by labels in set representation. If set, `nodeSelector` must not be set.

    - **spec.nodeLabelSelector.matchExpressions**
      array of objects

      List of label expressions that a node should have to qualify for the filter condition.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      - key: tier
        operator: NotIn
        values:
        - production
      ```

      - **spec.nodeLabelSelector.matchExpressions.key**
        string

        A label name.

      - **spec.nodeLabelSelector.matchExpressions.operator**
        string

        A comparison operator.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.nodeLabelSelector.matchExpressions.values**
          array of strings

          A label value.

            - **Element of the array**
              string

              Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

              Length: `1..63`
      - **spec.nodeLabelSelector.matchLabels**
        object
  - **spec.podLabelSelector**
    object

    Limiting the pods which are processed by labels.

      - **spec.podLabelSelector.matchExpressions**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

        ```
        matchExpressions:
        - key: tier
          operator: In
          values:
          - production
          - staging
        - key: tier
          operator: NotIn
          values:
          - production
        ```

          - **spec.podLabelSelector.matchExpressions.key**
            string

            A label name.

          - **spec.podLabelSelector.matchExpressions.operator**
            string

            A comparison operator.

            Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

          - **spec.podLabelSelector.matchExpressions.values**
            array of strings

            A label value.

              - **Element of the array**
                string

                Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

                Length: `1..63`
      - **spec.podLabelSelector.matchLabels**
        object
  - **spec.priorityClassThreshold**
    object

    Limiting the pods which are processed by priority class. Only pods under the threshold can be evicted.

    You can specify either the name of the priority class (priorityClassThreshold.name), or the actual value of the priority class (priorityThreshold.value).

    By default, this threshold is set to the value of `system-cluster-critical` priority class.

      - **spec.priorityClassThreshold.name**
        string

        Name of the priority class.

      - **spec.priorityClassThreshold.value**
        integer

        Value of the priority class.

  - **spec.protectedStorageClasses**
    array of strings

    A list of StorageClass names whose Pods must not be evicted.

    If a Pod refers to a PersistentVolumeClaim provisioned by one of the listed StorageClasses, the descheduler will never evict such a Pod.

    Note the following:

      - An empty list (or an omitted parameter) keeps the current behavior: Pods with PVCs are evicted as usual.
      - A Pod whose PersistentVolumeClaim cannot be resolved (the PVC is missing or has an empty `storageClassName`) is considered protected as well.

    Example:

    ```
    protectedStorageClasses:
    - local-path
    - local-storage-class
    ```

      - **Element of the array**
        string

        A StorageClass name.

        Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

        Length: `1..253`
  - **spec.strategies**
    object

    Required value

    Settings of strategies for the Descheduler instances.

      - **spec.strategies.highNodeUtilization**
        object

        The strategy evicts pods from under utilized nodes to concentrate workloads on fewer nodes. It is intended for use with node autoscaling and a scheduler using `MostAllocated`.

Node resource usage takes into account extended resources and is based on pod requests and limits, not actual consumption.

Detailed description of the strategy is provided in the HighNodeUtilization section.

- **spec.strategies.highNodeUtilization.enabled**
  boolean

  Makes the strategy active.

  Default: `false`

- **spec.strategies.highNodeUtilization.thresholds**
  object

  Sets threshold values to identify to identify under utilized nodes.

  If the resource usage of the node is below **all** threshold values, then the node is considered under utilized.

  If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

  - **spec.strategies.highNodeUtilization.thresholds.cpu**
    integer

    CPU fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.highNodeUtilization.thresholds.memory**
    integer

    Memory fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.highNodeUtilization.thresholds.pods**
    integer

    Pods count in percents

    Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization**
  object

  The strategy evicts pods from over utilized nodes, assuming they will be rescheduled on under utilized nodes.

  By default, a node is under utilized when its resource usage is below all values in `thresholds`, and over utilized when it exceeds at least one value in `targetThresholds`.

  Node resource usage takes into account extended resources and is based on pod requests and limits, not actual consumption.

  The utilization data source is determined by `metricsSource`.

  Detailed description of the strategy is provided in the LowNodeUtilization section.

  - **spec.strategies.lowNodeUtilization.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.lowNodeUtilization.evictionLimits**
    object

    Limits the number of pods the strategy evicts from one node.

    Example:

    ```
    evictionLimits:
      node: 2
    ```

    - **spec.strategies.lowNodeUtilization.evictionLimits.node**
      integer

      The maximum number of pods the strategy evicts from one node in one descheduling cycle.

      The limit also counts pods evicted in the background, such as a virtual machine pod that leaves the node through a live migration. Such a pod takes a place in the limit in every cycle until it leaves the node, and no new eviction request is sent for it. So for virtual machines the parameter effectively limits how many of them migrate from one node at the same time.

      If the parameter is not set, the strategy evicts pods from a node until the node stops being over utilized or the under utilized nodes run out of free resources.

      With metricsSource `Prometheus` the strategy evicts one pod from each over utilized node in a cycle without the parameter and up to the set number of pods with it, so in that mode the parameter raises the number of evictions instead of limiting it.

      Allowed values: `1 <= X`

  - **spec.strategies.lowNodeUtilization.metricsSource**
    string

    Source of node utilization data for the strategy.

    - `Default` — Pod resource requests or actual consumption from the Metrics API if the `metrics.k8s.io` API group is registered in the cluster.
    - `Prometheus` — node CPU utilization averaged over 15 minutes using cluster Prometheus data (node-exporter).

    With `Prometheus`, the `cpu` key is required in both thresholds and targetThresholds; other keys are ignored. The `prometheus` module must be enabled. Otherwise, the strategy uses `Default`, and the Deckhouse log shows a warning.

    Together with useDeviationThresholds, the strategy keeps nodes within a corridor around the 15-minute average CPU utilization of the cluster. If evictionLimits.node is set, the parameter increases the number of evictions in this mode instead of limiting it.

    If Prometheus is unavailable or has no data for at least one Ready node, the strategy skips balancing in that cycle.

    Detailed description of this mode is provided in the CPU utilization section.

    Default: `Default`

    Allowed values: `Default`, `Prometheus`

  - **spec.strategies.lowNodeUtilization.targetThresholds**
    object

    Sets threshold values to identify to identify over utilized nodes.

    If the resource usage of the node exceeds at least one of the threshold values, then the node is considered over utilized.

    If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

    - **spec.strategies.lowNodeUtilization.targetThresholds.cpu**
      integer

      CPU fraction in percents

      Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.memory**
  integer

  Memory fraction in percents

  Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.pods**
  integer

  Pods count in percents

  Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.thresholds**
  object

  Sets threshold values to identify to identify under utilized nodes.

  If the resource usage of the node is below **all** threshold values, then the node is considered under utilized.

  If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

  - **spec.strategies.lowNodeUtilization.thresholds.cpu**
    integer

    CPU fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.lowNodeUtilization.thresholds.memory**
    integer

    Memory fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.lowNodeUtilization.thresholds.pods**
    integer

    Pods count in percents

    Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.useDeviationThresholds**
  boolean

  Counts thresholds and targetThresholds from the average utilization of the nodes instead of using them as absolute values.

  The average is calculated over all Ready nodes of the cluster for every resource listed in `thresholds`, regardless of nodeLabelSelector.

  A node is considered under utilized if its usage of **all** listed resources is at or below the average minus the `thresholds` value. A node is considered over utilized if its usage of **at least one** listed resource is above the average plus the `targetThresholds` value. The resulting bounds are limited to the range from 0 to 100%.

  In this mode, a `thresholds` value may be greater than the `targetThresholds` value of the same resource. Both sections must still list the same resources.

  For example, `thresholds: {cpu: 5}` and `targetThresholds: {cpu: 5}` keep the CPU utilization of the nodes within the corridor of ±5% around the average.

  Default: `false`

- **spec.strategies.removeDuplicates**
  object

  The strategy ensures that no more than one pod of a ReplicaSet, ReplicationController, StatefulSet, or pods of a single Job is running on the same node. If there are two or more such pods, the module evicts the excess pods so that they are better distributed across the cluster.

  - **spec.strategies.removeDuplicates.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

- **spec.strategies.removePodsHavingTooManyRestarts**
  object

  The strategy evicts pods having too many restarts from nodes.

  Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold.

  This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

  - **spec.strategies.removePodsHavingTooManyRestarts.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.removePodsHavingTooManyRestarts.podRestartThreshold**
    integer

    Threshold for the total number of restarts across all pod containers, including init containers.

    If the number of restarts exceeds this value, the pod is considered unhealthy and becomes a candidate for eviction.

    Default: `100`

    Allowed values: `1 <= X`

- **spec.strategies.removePodsViolatingInterPodAntiAffinity**
  object

  The strategy ensures that pods violating inter-pod affinity and anti-affinity rules are evicted from nodes.

  - **spec.strategies.removePodsViolatingInterPodAntiAffinity.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

- **spec.strategies.removePodsViolatingNodeAffinity**
  object

  The strategy evicts pods that violate node affinity rules according to the configured nodeAffinityType.

  - **spec.strategies.removePodsViolatingNodeAffinity.enabled**
    boolean

    Makes the strategy active.

Default: `false`

- **spec.strategies.removePodsViolatingNodeAffinity.nodeAffinityType**
  array of strings

  Defines the list of node affinity rules used.

  Default: `["requiredDuringSchedulingIgnoredDuringExecution"]`

  - **Element of the array**
    string

    Allowed values: `requiredDuringSchedulingIgnoredDuringExecution`, `preferredDuringSchedulingIgnoredDuringExecution`

- **spec.strategies.removePodsViolatingTopologySpreadConstraint**
  object

  The strategy ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint's `maxSkew`.

  This is useful for rebalancing pods across zones after a zone outage recovery.

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.constraints**
    array of strings

    List of topology spread constraint types to consider when deciding which pods to evict.

    Topology spread constraints control how pods are distributed across your cluster topology (e.g. regions, zones, nodes).

    By default, only `DoNotSchedule` (hard constraints) are considered — these are constraints that the scheduler strictly enforces: a pod will NOT be placed if doing so would violate the constraint. `Descheduler` will evict pods that are currently violating such constraints (e.g. due to node failures or cluster topology changes after initial scheduling).

    Optionally, `ScheduleAnyway` (soft constraints) can also be included — these are constraints that the scheduler tries to satisfy but does not strictly enforce: a pod WILL be placed even if the constraint cannot be satisfied, though the scheduler will attempt to minimize the imbalance. Including this type causes `descheduler` to also evict pods that violate soft constraints.

    Default: `["DoNotSchedule"]`

    - **Element of the array**
      string

      Allowed values: `DoNotSchedule`, `ScheduleAnyway`

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.topologyBalanceNodeFit**
    boolean

    If set to `true`, the strategy will consider whether the pods can fit on other nodes when balancing topology domains.

    Default: `true`

Deprecated resource. Support for the resource might be removed in a later release.

Descheduler is a description of a single descheduler instance.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
  - **metadata.name**
    string
- **spec**
  object

  Required value

  Defines the behavior of a descheduler instance.

  - **spec.deploymentTemplate**
    object

    Defines Template of a descheduler Deployment.

    - **spec.deploymentTemplate.nodeSelector**
      object
    - **spec.deploymentTemplate.tolerations**
      array of objects
      - **spec.deploymentTemplate.tolerations.effect**
        string
      - **spec.deploymentTemplate.tolerations.key**
        string
      - **spec.deploymentTemplate.tolerations.operator**
        string
      - **spec.deploymentTemplate.tolerations.tolerationSeconds**
        integer
      - **spec.deploymentTemplate.tolerations.value**
        string
  - **spec.deschedulerPolicy**
    object

    `globalParameters` and `strategies` follow the descheduler's documentation.

    - **spec.deschedulerPolicy.globalParameters**
      object

      Parameters that apply to all policies.

      - **spec.deschedulerPolicy.globalParameters.evictFailedBarePods**
        boolean

        Allows Pods without `ownerReferences` and in `failed` phase to be evicted.

      - **spec.deschedulerPolicy.globalParameters.evictLocalStoragePods**
        boolean

        Allows Pods using local storage to be evicted.

      - **spec.deschedulerPolicy.globalParameters.evictSystemCriticalPods**
        boolean

        Allows eviction of Pods of any priority (including Kubernetes system Pods).

      - **spec.deschedulerPolicy.globalParameters.ignorePvcPods**
        boolean

        Prevents Pods with PVCs from being evicted.

- **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNamespace**
  integer

  Restricts maximum of Pods to be evicted per namespace.

- **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNode**
  integer

  Restricts maximum of Pods to be evicted per node.

- **spec.deschedulerPolicy.globalParameters.nodeSelector**
  string

- **spec.deschedulerPolicy.strategies**
  object

  List of strategies with corresponding parameters for a given Descheduler instances.

  - **spec.deschedulerPolicy.strategies.highNodeUtilization**
    object

    This strategy finds nodes that are under utilized and evicts Pods from the nodes in the hope that these Pods will be scheduled compactly into fewer nodes.

    - **spec.deschedulerPolicy.strategies.highNodeUtilization.enabled**
      boolean

      Required value

    - **spec.deschedulerPolicy.strategies.highNodeUtilization.namespaceFilter**
      object

      Restricts Namespaces to which this strategy applies.

    - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter**
      object

      Filters Nodes to which the strategy applies.

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector**
        object
        - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector.matchExpressions**
          array of objects

          List of label expressions that a node should have to qualify for the filter condition.

          Example:

          ```
          matchExpressions:
          - key: tier
            operator: In
            values:
            - production
            - staging
          - key: tier
            operator: NotIn
            values:
            - production
          ```

          - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector.matchExpressions.key**
            string

            A label name.

          - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector.matchExpressions.operator**
            string

            A comparison operator.

            Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

          - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector.matchExpressions.values**
            array of strings

            A label value.

            - **Element of the array**
              string

              Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

              Length: `1..63`

        - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter.labelSelector.matchLabels**
          object

    - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFit**
      boolean

      If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

    - **spec.deschedulerPolicy.strategies.highNodeUtilization.priorityFilter**
      object

      Only Pods with priority lower than this will be descheduled.

  - **spec.deschedulerPolicy.strategies.lowNodeUtilization**
    object

    This strategy finds nodes that are under utilized and evicts Pods, if possible, from other nodes in the hope that recreation of evicted Pods will be scheduled on these underutilized nodes.

    - **spec.deschedulerPolicy.strategies.lowNodeUtilization.enabled**
      boolean

      Required value

    - **spec.deschedulerPolicy.strategies.lowNodeUtilization.namespaceFilter**
      object

      Restricts Namespaces to which this strategy applies.

    - **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter**
      object

      Filters Nodes to which the strategy applies.

      - **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector**
        object
        - **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector.matchExpressions**
          array of objects

          List of label expressions that a node should have to qualify for the filter condition.

          Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector.matchExpressions.key**
  string

  A label name.

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector.matchExpressions.operator**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector.matchExpressions.values**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter.labelSelector.matchLabels**
  object

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFit**
  boolean

  If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removeDuplicates**
  object

  This strategy makes sure that there is only one Pod associated with a ReplicaSet (RS), ReplicationController (RC), StatefulSet, or Job running on the same node.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removeDuplicates.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

    - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector**
      object
      - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector.matchExpressions**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

        ```
        matchExpressions:
        - key: tier
          operator: In
          values:
          - production
          - staging
        - key: tier
          operator: NotIn
          values:
          - production
        ```

        - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector.matchExpressions.key**
          string

          A label name.

        - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector.matchExpressions.operator**
          string

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector.matchExpressions.values**
          array of strings

          A label value.

          - **Element of the array**
            string

            Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

            Length: `1..63`

      - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter.labelSelector.matchLabels**
        object

    - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFit**
      boolean

      If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.removeDuplicates.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removeFailedPods**
  object

  This strategy evicts Pods that are in `failed` status phase.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removeFailedPods.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

    - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector**
      object
      - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector.matchExpressions**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

        ```
        matchExpressions:
        - key: tier
          operator: In
          values:
          - production
          - staging
        - key: tier
          operator: NotIn
          values:
          - production
        ```

        - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector.matchExpressions.key**
          string

          A label name.

        - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector.matchExpressions.operator**
          string

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector.matchExpressions.values**
          array of strings

          A label value.

          - **Element of the array**
            string

            Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

            Length: `1..63`

      - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter.labelSelector.matchLabels**
        object

  - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts**
  object

  This strategy makes sure that Pods having too many restarts are removed from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

    - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector**
      object
      - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector.matchExpressions**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector.matchExpressio**
  string

  A label name.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector.matchExpressio**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector.matchExpressio**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter.labelSelector.matchLabels**
  object

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFit**
  boolean

  If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity**
  object

  This strategy makes sure that Pods violating interpod anti-affinity are removed from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

    - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector**
      object
      - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector.matchExpressio**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

        - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector.matchExpr**
          string

          A label name.

        - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector.matchExpr**
          string

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector.matchExpr**
          array of strings

          A label value.

          - **Element of the array**
            string

            Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

            Length: `1..63`

      - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter.labelSelector.matchLabels**
        object

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.priorityFilter**

object

Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity**
object

This strategy makes sure all Pods violating node affinity are eventually removed from nodes.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.enabled**
boolean

Required value

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.namespaceFilter**
object

Restricts Namespaces to which this strategy applies.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter**
object

Filters Nodes to which the strategy applies.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector**
object
- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector.matchExpressions**
array of objects

List of label expressions that a node should have to qualify for the filter condition.

Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector.matchExpressions.**
string

A label name.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector.matchExpressions.**
string

A comparison operator.

Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector.matchExpressions.**
array of strings

A label value.

- **Element of the array**
string

Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Length: `1..63`

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter.labelSelector.matchLabels**
object

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFit**
boolean

If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.priorityFilter**
object

Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints**
object

This strategy makes sure that Pods violating `NoSchedule` taints on nodes are removed.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.enabled**
boolean

Required value

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.namespaceFilter**
object

Restricts Namespaces to which this strategy applies.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter**
object

Filters Nodes to which the strategy applies.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector**
object
- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector.matchExpressions**
array of objects

List of label expressions that a node should have to qualify for the filter condition.

Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector.matchExpressions.k**
  string

  A label name.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector.matchExpressions.o**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector.matchExpressions.v**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter.labelSelector.matchLabels**
  object

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFit**
  boolean

  If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint**
  object

  This strategy makes sure that Pods violating topology spread constraints are evicted from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

    - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector**
      object
      - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector.matchExp**
        array of objects

        List of label expressions that a node should have to qualify for the filter condition.

        Example:

        ```
        matchExpressions:
        - key: tier
          operator: In
          values:
          - production
          - staging
        - key: tier
          operator: NotIn
          values:
          - production
        ```

        - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector.matc**
          string

          A label name.

        - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector.matc**
          string

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector.matc**
          array of strings

          A label value.

          - **Element of the array**
            string

            Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

            Length: `1..63`

      - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter.labelSelector.matchLab**
        object

  - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **status**
  object

  Most recently observed status of a descheduler instance.

  - **status.ready**
    boolean

    Required value

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Configuring the pod redistribution interval

To set how often the `descheduler` module runs the pod redistribution cycle, use the `deschedulingInterval` parameter.

For example, to run `descheduler` every 5 minutes, set `deschedulingInterval: Frequent`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true
  settings:
    deschedulingInterval: Frequent
```

Supported values:

- `Frequent`: Runs every 5 minutes. Suitable for clusters where faster pod redistribution is important.
- `Moderate`: Runs every 15 minutes (default).
- `Rare`: Runs every 30 minutes. Suitable for clusters where it's important to minimize the number of pod redistributions.

Example LowNodeUtilization strategy

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

Example LowNodeUtilization strategy with thresholds relative to the average

The strategy keeps the CPU utilization of the nodes within the corridor of ±5% around the average utilization of the cluster nodes and evicts at most two pods from one node per descheduling cycle. A virtual machine that is still migrating from the node takes one of the two places:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization-deviation
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      useDeviationThresholds: true
      thresholds:
        cpu: 5
      targetThresholds:
        cpu: 5
      evictionLimits:
        node: 2
```

Example LowNodeUtilization strategy on the 15-minute CPU average

The strategy keeps the nodes within the corridor of ±5% around the cluster average of the CPU utilization averaged over 15 minutes, so short load spikes do not trigger evictions. It evicts one pod from each over utilized node per descheduling cycle; keep `evictionLimits.node` unset in this mode, since it raises that number. Use the default `Moderate` descheduling interval (15 minutes). The `prometheus` module must be enabled:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization-prometheus
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      metricsSource: Prometheus
      useDeviationThresholds: true
      thresholds:
        cpu: 5
      targetThresholds:
        cpu: 5
```

Example HighNodeUtilization strategy

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

Protecting pods by StorageClass

Use the `protectedStorageClasses` parameter to forbid evicting pods that use PersistentVolumeClaims provisioned by specific StorageClasses. This is useful for storage that is bound to a node (for example, `local-path`), where evicting a pod either loses the data locality or leaves the pod unschedulable.

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: protect-local-storage
spec:
  protectedStorageClasses:
    - local-path
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

With this configuration, the descheduler can still evict every other pod, but a pod with a PersistentVolumeClaim from the `local-path` StorageClass is never evicted.

Keep the following in mind:

- If the parameter is omitted or empty, the behavior does not change: pods with PersistentVolumeClaims are evicted as usual.
- A pod is also considered protected when its PersistentVolumeClaim cannot be resolved, that is, the PVC does not exist or its `storageClassName` is empty. Such pods are never evicted while the parameter is set.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module documentation

> The module lifecycle stage: General Availability

> Web interfaces associated with the module: documentation

The `documentation` module creates a web interface for access to the documentation of the Deckhouse Platform version running in the cluster and to the documentation of the available modules.

To get the web interface address, in the publicDomainTemplate template of the Deckhouse Platform global configuration parameter, replace the `%s` key with `documentation`. For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, then the documentation web interface will be available at `documentation-kube.company.my`.

Main features

The module provides the following features:

- The documentation of the Deckhouse Platform version running in the cluster, without access to the Internet
- The documentation of the modules of every ModuleSource of the cluster, enabled or not
- Search throughout the platform documentation and the modules
- The pages in Markdown, an `llms.txt` index and a `corpus.json` file for AI tools
- Authentication through the `user-authn` module, an external authentication service or basic authentication with a generated password: see the usage page
- Keeping the prepared versions in the system S3 storage or in an S3 storage of your own, and a high availability mode with two replicas

How the module works

The documentation of the Deckhouse Platform version the module is part of is built into the module.

The documentation of a specific module version is prepared on demand and cached:

1. A user opens a page of a module version that is not prepared yet.
2. The module downloads the documentation from the image of the version in the container registry and builds its pages.
3. The prepared version is kept on the volume of the documentation pod. When keeping versions in S3 is on, the version is kept in S3 too, and after a restart of the pod the module opens it without preparing it again.
4. One version of every module is kept always. For an enabled module it is the installed version. For the other modules it is the version of the most stable channel: stable when it exists. The search index and the AI tool files (`llms.txt`, `corpus.json`) of all modules are built from these versions.

When a module has a ModuleRelease in the `Pending` phase, the module prepares the documentation of that version in advance. The module reads the list of versions from the container registry every 30 minutes. A version published later shows up in the list within 30 minutes.

Limitations

Each documentation pod keeps the prepared versions on a 4 GiB volume, in two parts of 2 GiB:

Part	Content	Room	Removal
The versions users open	A version takes about 1.3 MB	About 1500 versions	Above 2 GiB the version opened longest ago goes first; a version not opened for 7 days goes anyway and is prepared again when opened
The versions the search uses	One version of each module, with its search and AI files; about 3.5 MB each	About 600 versions	Never removed; the `D8DocumentationPinnedVersionsTooLarge` alert fires when they take more than 2 GiB

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The documentation module: configuration

> The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

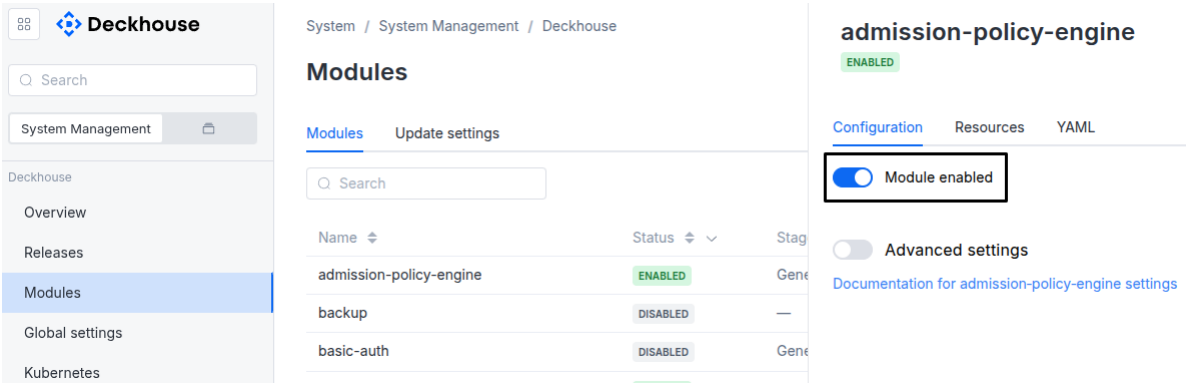
The module has 9 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `documentation` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable documentation
```

- **Using ModuleConfig `documentation`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `documentation` (create it if necessary);

  Example of a manifest to enable module `documentation`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  enabled: true
```

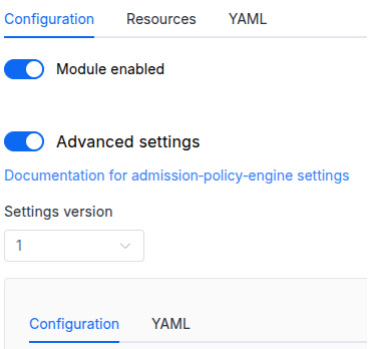
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `documentation` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `documentation` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `documentation`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `documentation` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc documentation
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `documentation` using the example below. Fill in the `spec.settings` section with the required module

parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `documentation`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.auth**
    object

    Parameters to authenticate and authorize access to the documentation web interface.

    - **settings.auth.allowedUserEmails**
      array of strings

      Array of emails of users that can access module's public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      Array of groups whose users can browse the documentation.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

      > **Caution!** If the module works together with the `user-authn` module, add the allowed groups to the DexProvider configuration.

      Default: `[]`

      Examples:

      ```
      allowedUserGroups:
      - admin
      - users
      ```

      ```
      allowedUserGroups: []
      ```

    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

      > External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response).

        Default:

        Example:

        ```
        authSignInURL: https://$host/dex-authenticator/sign_in
        ```

      - **settings.auth.externalAuthentication.authURL**
        string

        URL of the authentication service.

        If the user is authenticated, the service should return an HTTP 200 response code.

        Default:

        Example:

        ```
        authURL: https://documentation-dex-authenticator.d8-system.svc.cluster.local/dex-authenticator/auth
        ```

  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse Platform automatically decides whether to enable the HA mode. Learn more about the high availability mode for modules.

    In the HA mode the documentation runs in two replicas on different nodes; the "High availability mode" section of the usage page of the module tells what it needs and why S3 helps.

    Examples:

    ```
    highAvailability: true
    ```

    ```
    highAvailability: false
    ```

  - **settings.https**
    object

    What certificate type to use.

    This parameter completely overrides the `global.modules.https` settings.

    Examples:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**
  object

  Parameters for certmanager.

  - **settings.https.certManager.clusterIssuerName**
    string

    What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

    Default: `letsencrypt`

    Example:

    ```
    clusterIssuerName: letsencrypt
    ```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    Name of the Secret in the `d8-system` namespace to use with the documentation web UI.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  HTTPS usage mode:

  - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace.
  - `Disabled` — in this mode, the documentation web UI can only be accessed over HTTP.
  - `OnlyInURI` — HTTP access with an external HTTPS balancer. The balancer terminates HTTPS, and all links in `user-authn` are generated with the HTTPS scheme. The balancer must provide redirection from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  Class of the Ingress controller of the documentation web UI.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.moduleStorage**
  object

  Storage of the documentation of module versions, prepared when a version is opened for the first time.

  - **settings.moduleStorage.s3**
    object

    Where the prepared documentation of module versions is kept besides the cache of the documentation pod.

    - **settings.moduleStorage.s3.external**
      object

      S3 storage of your own for `type: External`. The connection and the keys are taken from a Secret; the keys it holds are listed in the “S3 storage of your own” section of the usage page of the module.

      - **settings.moduleStorage.s3.external.prefix**
        string

        Part of the bucket the module uses: every object it reads, writes or deletes has a key starting with this prefix. Other objects of the bucket stay untouched, so the bucket may be shared.

        Default: `documentation/`

        Pattern: `^([A-Za-z0-9][A-Za-z0-9._-]*/)+$`

        Examples:

        ```
        prefix: documentation/
        ```

        ```
        prefix: clusters/main/documentation/
        ```

      - **settings.moduleStorage.s3.external.secretRef**
        object

        Required value

        Secret with the connection and the keys of the S3 storage, in any namespace.

        Deckhouse Platform copies it into the `documentation-s3-connection` Secret in `d8-system`. The documentation pod uses that Secret. After the Secret is created or changed, the documentation pod restarts within 5 minutes. Until the Secret appears, the versions are kept in the pod only.

        - **settings.moduleStorage.s3.external.secretRef.name**
          string

          Required value

Name of the Secret.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Maximum length: `253`

Example:

```
name: documentation-s3
```

- **settings.moduleStorage.s3.external.secretRef.namespace**
  string

  Required value

  Namespace of the Secret.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Maximum length: `63`

  Examples:

  ```
  namespace: d8-system
  ```

  ```
  namespace: documentation-storage
  ```

- **settings.moduleStorage.s3.external.size**
  string

  How much of the bucket the module may take under the prefix. Above 80% of it the module deletes stored versions until 60% is left, in this order:

  1. Copies replaced after an update of the documentation template.
  2. Versions opened by readers.
  3. Versions used by the search.

  Within each group the versions stored longest ago go first.

  Default: `5Gi`

  Pattern: `^[1-9][0-9]*(Mi|Gi|Ti)$`

  Examples:

  ```
  size: 5Gi
  ```

  ```
  size: 20Gi
  ```

- **settings.moduleStorage.s3.type**
  string
  - `Disabled` — only in the pod: after a restart of the pod the versions are prepared again when opened. In the high availability mode each replica prepares and keeps the versions on its own, with twice the CPU and the requests to the registry; turn S3 on for that mode.
  - `System` — also in the system S3 storage of the system-s3 module: after a restart of the pod the versions open from it in a fraction of a second. The module creates a 5 GiB bucket for it. Requires the `system-s3` module to be enabled: without it the versions are kept in the pod only, and the `D8DocumentationSystemS3Disabled` alert fires.
  - `External` — also in an S3 storage of your own, set in the `external` section.

  Default: `Disabled`

  Allowed values: `Disabled`, `System`, `External`

- **settings.nodeSelector**
  object

  Same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.tolerations**
  array of objects

  Same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The documentation module: usage

> The module lifecycle stage: General Availability

An example of the module configuration

The module runs on the system nodes, authenticates users through an external authentication service and lets in the users of the `admins` group:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
    auth:
      externalAuthentication:
        authURL: "https://<applicationDomain>/auth"
        authSignInURL: "https://<applicationDomain>/sign-in"
      allowedUserGroups:
      - admins
```

Authentication

The user-authn module provides authentication by default. Also, an external authentication service can be set in the auth.externalAuthentication parameter. When the `user-authn` module is disabled and the `auth.externalAuthentication` parameter is not set, the module uses basic authentication with the user `admin` and an auto-generated password.

Use `d8 k` to see the password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values documentation -o json | jq '.internal.auth.password'
```

Delete the Secret to re-generate the password:

```
d8 k -n d8-system delete secret/documentation-basic-auth
```

S3 storage of your own

With `moduleStorage.s3.type: External` the module keeps the prepared documentation of module versions in an S3 bucket of your own. Create a Secret in any namespace with these keys. Deckhouse Platform copies it into the `documentation-s3-connection` Secret in `d8-system`. After the Secret is created or changed, the documentation pod restarts within 5 minutes. Keep only the connection keys in the Secret: all of its keys are copied.

Key	Required	Value
`S3_ENDPOINT`	yes	The address of the storage, `https://` only.
`S3_BUCKET`	yes	The name of the bucket.
`S3_REGION`	yes	The region of the bucket, for example `us-east-1`.
`AWS_ACCESS_KEY_ID`	yes	The access key.
`AWS_SECRET_ACCESS_KEY`	yes	The secret key.
`S3_CA_CERT`	no	The CA certificate of the storage in PEM format, when the system ones do not trust it.

The key needs the rights to read, write, list and delete objects under the prefix from moduleStorage.s3.external.prefix (`documentation/` by default). The module reads, writes and deletes no object outside of the prefix, so the S3 bucket may be shared with other data.

An example of the Secret and of the module configuration that uses it:

```
apiVersion: v1
kind: Secret
metadata:
  name: documentation-s3
  namespace: documentation-storage
stringData:
  S3_ENDPOINT: https://s3.example.com
  S3_BUCKET: documentation
  S3_REGION: us-east-1
  AWS_ACCESS_KEY_ID: <access key>
  AWS_SECRET_ACCESS_KEY: <secret key>
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  settings:
    moduleStorage:
      s3:
        type: External
        external:
          secretRef:
            namespace: documentation-storage
            name: documentation-s3
```

While the Secret does not exist or the storage does not answer, the versions are kept in the pod only. After a restart of the pod the module prepares them again. The `docs_builder ondemand_s3_ready` metric of the documentation pod is 0 then. A mistake in the Secret (a missing or misnamed key, an `http://` address) is written to the log of the `builder` container of the documentation pod. When the storage stays not ready for 15 minutes, the `D8DocumentationS3NotReady` alert fires.

High availability mode

The high availability mode is turned on by the highAvailability parameter; when the parameter is not set, the HA mode of the cluster applies. In this mode the documentation runs in two replicas on different nodes, so two nodes the module can run on are needed (the system nodes by default). With one such node the second replica stays `Pending`, and the PodDisruptionBudget blocks draining the node with the running replica until the second replica runs on another node. The authenticator of the documentation (DexAuthenticator), if it is deployed, also runs in two replicas.

A user stays on one replica: the `d8-documentation-affinity` cookie picks it. Through an Ingress the cookie lives 24 hours, also after the browser is closed; through the Gateway API it lives until the browser is closed. That replica prepares and keeps the module versions the user opens.

Turn on keeping versions in S3 (moduleStorage.s3.type: `System` or `External`): then a version prepared by one replica is opened by the other from S3 without

preparing it again, and a user moved to the other replica when theirs restarts does not wait for the versions again. Without S3 each replica reads the container registry on its own and prepares on its own the versions the search uses and the versions its users open. In this case the documentation takes twice the CPU and sends twice the requests to the registry, and a user moved to the other replica waits for a version they already opened once more.

With S3 one replica, picked through a Lease, reads the container registry and prepares the versions the search uses; the other takes them from S3 within a minute and prepares only the versions its users open. A replica starts taking users once it has received the versions from S3, at most 5 minutes after its start.

After an update of Deckhouse Platform that changes the look of the documentation, the versions kept in S3 open as they were and are prepared again in the background, one at a time, when no user waits for a version. A version that the new look can not prepare stays as it was.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The infra-provider-baremetal module

The module lifecycle stage: Experimental

The module has requirements for installation

Before enabling the module, you need to allow the use of experimental modules in the `deckhouse` ModuleConfig. To do this, set the allowExperimentalModules parameter to `true`.

The `infra-provider-baremetal` module allows `node-manager` to create CloudEphemeral nodes on physical servers. It manages servers through their baseboard management controllers (BMCs), installs an operating system on them over the network, and joins them to the cluster. The module is based on components of the Metal3 project and the Ironic service.

Key features

The module allows you to perform the following tasks:

- Register physical servers by BMC address and credentials with automatic selection of the Redfish or IPMI protocol
- Create and delete nodes in node groups with the `CloudEphemeral` type, including scaling groups from zero with cluster autoscaler
- Install an operating system image on servers and clean disk metadata when a server is released
- Deploy Ironic in the cluster or connect to an existing Ironic instance
- Protect server boot traffic with TLS

How it works

A node on a physical server appears in the cluster in the following order:

1. The administrator describes the server in a BareMetalInstance resource. The module connects to the BMC, finds the server by its UUID, and creates a Metal3 BareMetalHost resource for it.
2. Ironic boots the server over the network, collects hardware information, and moves the server to the `available` state.
3. When a node group with a BareMetalInstanceClass instance class needs a new node, Cluster API picks a free server by labels, and Ironic writes the image from the BareMetalImage resource to its disk.
4. The server boots the installed operating system, runs the initial configuration script, and joins the cluster as a node of the group.
5. When the node is deleted, Ironic cleans the server disk metadata by default, and the server becomes available for new nodes again.

Requirements

The module requires the following:

- A static cluster with no `cloud-provider-*` cloud provider modules enabled
- Servers whose BMC supports the Redfish or IPMI protocol and is reachable over the network from the cluster
- A provisioning network where servers boot over PXE and cluster nodes connected to it where Ironic can start if it runs in the cluster
- An operating system image with cloud-init and `python3`, where cloud-init configures network interfaces by itself, for example, over DHCP
- A storage module whose CSI driver runs on the created nodes

A storage module is required because Deckhouse Platform does not place workloads on a new node until a CSI driver registers on it.

Limitations

The module has the following limitations:

- Node groups use a single `default` zone.
- A BMC CA certificate in the `spec.bmc.tls.caCertRef` parameter can be set only if Ironic runs in the cluster.
- TLS for server boot is available only if Ironic runs in the cluster and uses the built-in DHCP server.

Third-party components

List of third-party software used in the `infra-provider-baremetal` module:

- Cluster API Provider Metal3 v1.9.5

  License: Apache License 2.0

  Infrastructure provider for Kubernetes Cluster API backed by Metal3.

- Metal3 Bare Metal Operator v0.13.1

  License: Apache License 2.0

  Kubernetes operator for provisioning bare metal hosts with Ironic.

- Ironic Standalone Operator v0.10.0

  License: Apache License 2.0

  Kubernetes operator managing standalone Ironic deployments for Metal3.

- Metal3 Ironic Image v39.0.0

  License: Apache License 2.0

  Container image to run OpenStack Ironic as part of Metal3.

- OpenStack Ironic 752100ac7ff8774510a34963839aaed93bbe8589

  License: Apache License 2.0

  Bare metal provisioning service.

- OpenStack Ironic Python package 39.0.0

  License: Apache License 2.0

  Python distribution version of the pinned Ironic source commit.

- Ironic Python Agent ramdisk ipa-centos9-stable-2026.1

  License: Apache License 2.0

  Default IPA deployment kernel and initramfs.

- pip 25.3

License: MIT

Python package installer used during the Ironic build.

- setuptools 84.2.1

  License: MIT

  Python packaging tools used during the Ironic build.

- wheel 0.38.4

  License: MIT

  Python wheel builder used during the Ironic build.

- pbr 7.1.3

  License: Apache License 2.0

  OpenStack Python build backend used for Ironic.

- Python wheels 0.19

  License: MIT

  Python dependency wheels supplied by the Deckhouse package manager.

- RPM rpm-4.20.1-release

  License: GPL-2.0-or-later

  Upstream rpm2cpio script used to extract signed EFI boot loaders during the Ironic build.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: Experimental
>
> The module has requirements for installation
>
> Before enabling the module, you need to allow the use of experimental modules in the `deckhouse` ModuleConfig. To do this, set the allowExperimentalModules parameter to `true`.

The module is disabled by default. To enable it, create the `infra-provider-baremetal` ModuleConfig and set the `nodes.parameters.ironic` parameter in it. Until the parameter is set, the module components are not started.

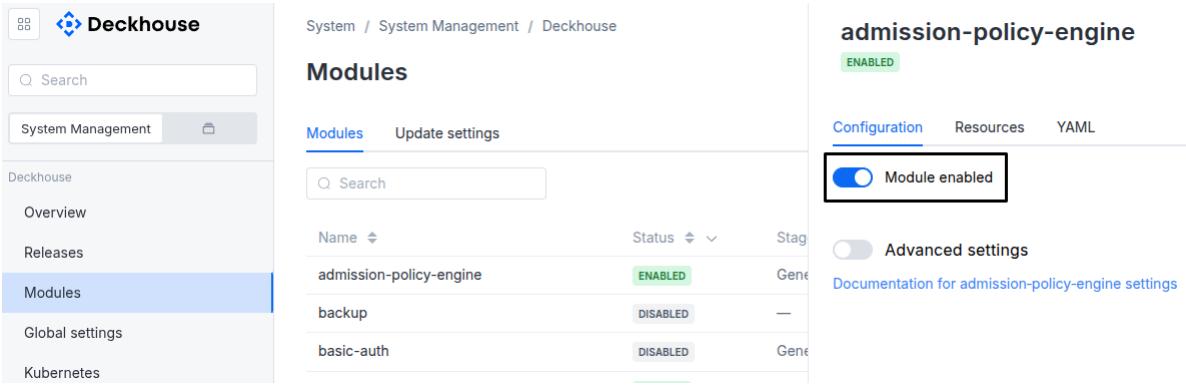
Choosing the Ironic mode and the setup procedure are described on the “Usage” page.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `infra-provider-baremetal` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable infra-provider-baremetal
```

- **Using ModuleConfig `infra-provider-baremetal`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `infra-provider-baremetal` (create it if necessary);

  Example of a manifest to enable module `infra-provider-baremetal`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: infra-provider-baremetal
spec:
  enabled: true
```

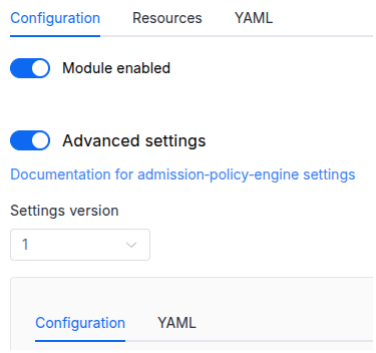
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `infra-provider-baremetal` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `infra-provider-baremetal` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `infra-provider-baremetal`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `infra-provider-baremetal` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc infra-provider-baremetal
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `infra-provider-baremetal` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `infra-provider-baremetal`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: infra-provider-baremetal
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the Deckhouse version: 1.68.0 and above.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.nodes**
    object

    Settings for creating cluster nodes on physical servers.

    - **settings.nodes.parameters**
      object

      Required value

      Parameters for creating cluster nodes on physical servers.

      - **settings.nodes.parameters.ironic**
        object

        Required value

        Settings of the Ironic service that the module uses to manage servers and install the operating system on them.

        To connect to Ironic deployed outside the cluster, set the `externalInstance` parameter. To deploy Ironic in the cluster, set the `provisioningNetwork` and `dhcp` parameters.

        - **settings.nodes.parameters.ironic.bmcProbeTimeoutSeconds**
          integer

          Timeout of a single protocol check when connecting to a baseboard management controller (BMC), in seconds.

          Default: `15`

          Allowed values: `1 <= X <= 300`
        - **settings.nodes.parameters.ironic.dhcp**
          object

          DHCP settings in the provisioning network.

          Set either the `internal` or the `external` parameter.

          - **settings.nodes.parameters.ironic.dhcp.external**
            object

            Settings of an external DHCP server in the provisioning network.

            The module does not deploy a DHCP server and does not configure the external one. Configure PXE boot of servers on the external DHCP server yourself.

            - **settings.nodes.parameters.ironic.dhcp.external.pxeBootFile**
              object

              Boot file names configured on the external DHCP server.

              - **settings.nodes.parameters.ironic.dhcp.external.pxeBootFile.bios**
                string

Boot file name for servers in legacy BIOS mode.

Length: `1..1024`

- **settings.nodes.parameters.ironic.dhcp.external.pxeBootFile.uefi**
  string

  Boot file name for servers in UEFI mode.

  Length: `1..1024`

- **settings.nodes.parameters.ironic.dhcp.external.pxeBootServer**
  string

  Required value

  IPv4 address of the PXE server that the external DHCP server passes to servers.

- **settings.nodes.parameters.ironic.dhcp.internal**
  object

  Settings of the built-in DHCP server that Ironic runs in the provisioning network.

  - **settings.nodes.parameters.ironic.dhcp.internal.disableDNS**
    boolean

    Disables the built-in DNS server and prevents DHCP from passing a DNS server address to servers.

    By default, DHCP passes `dnsAddress` if specified. Otherwise, Ironic serves DNS and forwards queries using the resolver configuration of its pod.

    Set to `true` for an isolated provisioning network that does not need DNS.

    When `true`, cannot be used with the `dnsAddress` parameter.

    Default: `false`

  - **settings.nodes.parameters.ironic.dhcp.internal.dnsAddress**
    string

    DNS server address that the DHCP server passes to servers.

    Cannot be used if the `disableDNS` parameter is `true`.

  - **settings.nodes.parameters.ironic.dhcp.internal.gatewayAddress**
    string

    Default gateway address that the DHCP server passes to servers.

  - **settings.nodes.parameters.ironic.dhcp.internal.networkCIDR**
    string

    Required value

    Provisioning network subnet in CIDR format.

  - **settings.nodes.parameters.ironic.dhcp.internal.rangeBegin**
    string

    Required value

    First IPv4 address of the range the DHCP server assigns to servers.

  - **settings.nodes.parameters.ironic.dhcp.internal.rangeEnd**
    string

    Required value

    Last IPv4 address of the range the DHCP server assigns to servers.

- **settings.nodes.parameters.ironic.externalInstance**
  object

  Settings for connecting to Ironic deployed outside the cluster.

  If the parameter is set, the module does not deploy Ironic or a DHCP server. Cannot be used with the `provisioningNetwork`, `dhcp`, `ramdiskImageRef`, `ipxeFirmwareRef`, and `provisioningTLS` parameters.

  - **settings.nodes.parameters.ironic.externalInstance.credentialsRef**
    object

    Reference to the Secret with credentials for the external Ironic API.

    The Secret must contain the `identity` key with the user name and the `secret` key with the password.

    - **settings.nodes.parameters.ironic.externalInstance.credentialsRef.kind**
      string

      Required value

      Kind of the resource with external Ironic API credentials.

      Allowed values: `Secret`

    - **settings.nodes.parameters.ironic.externalInstance.credentialsRef.name**
      string

      Required value

      Name of the Secret with external Ironic API credentials.

      Length: `1..253`

    - **settings.nodes.parameters.ironic.externalInstance.credentialsRef.namespace**
      string

      Required value

      Namespace of the Secret with external Ironic API credentials.

      Only the `d8-infra-provider-baremetal` namespace is supported.

      Pattern: `^d8-infra-provider-baremetal$`

  - **settings.nodes.parameters.ironic.externalInstance.endpoint**
    string

    Required value

    URL of the external Ironic API.

    The URL must end with `/v1` or `/v1/` and be reachable from the cluster.

    Pattern: `^https?://.+/v1/?$`

    Maximum length: `2048`

  - **settings.nodes.parameters.ironic.externalInstance.tls**

object

TLS settings for connecting to the external Ironic.

- **settings.nodes.parameters.ironic.externalInstance.tls.caCertRef**
  object

  Reference to the Secret key with the CA certificate that issued the external Ironic API certificate.

  - **settings.nodes.parameters.ironic.externalInstance.tls.caCertRef.key**
    string

    Required value

    Key in the Secret that holds the external Ironic CA certificate.

    Length: `1..253`
  - **settings.nodes.parameters.ironic.externalInstance.tls.caCertRef.kind**
    string

    Required value

    Kind of the resource with the external Ironic CA certificate.

    Allowed values: `Secret`
  - **settings.nodes.parameters.ironic.externalInstance.tls.caCertRef.name**
    string

    Required value

    Name of the Secret with the external Ironic CA certificate.

    Length: `1..253`
  - **settings.nodes.parameters.ironic.externalInstance.tls.caCertRef.namespace**
    string

    Required value

    Namespace of the Secret with the external Ironic CA certificate.

    Only the `d8-infra-provider-baremetal` namespace is supported.

    Pattern: `^d8-infra-provider-baremetal$`
- **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef**
  object

  Reference to the Secret with the client certificate and private key for mutual TLS authentication.

  - **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef.certKey**
    string

    Required value

    Key in the Secret that holds the client certificate.

    Length: `1..253`
  - **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef.kind**
    string

    Required value

    Kind of the resource with the client certificate.

    Allowed values: `Secret`
  - **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef.name**
    string

    Required value

    Name of the Secret with the client certificate and private key.

    Length: `1..253`
  - **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef.namespace**
    string

    Required value

    Namespace of the Secret with the client certificate and private key.

    Only the `d8-infra-provider-baremetal` namespace is supported.

    Pattern: `^d8-infra-provider-baremetal$`
  - **settings.nodes.parameters.ironic.externalInstance.tls.clientCertRef.privateKey**
    string

    Required value

    Key in the Secret that holds the client private key.

    Length: `1..253`
- **settings.nodes.parameters.ironic.externalInstance.tls.insecure**
  boolean

  Disables verification of the external Ironic API certificate.

  Do not use in production environments.

  Default: `false`

- **settings.nodes.parameters.ironic.ipxeFirmwareRef**
  object

  Reference to the BareMetalIPXEFirmware resource with iPXE firmware for server boot over TLS.

  Used only together with the `provisioningTLS` parameter. If the parameter is set without `provisioningTLS`, the module does not create Ironic.

  - **settings.nodes.parameters.ironic.ipxeFirmwareRef.kind**
    string

    Required value

    Kind of the resource with iPXE firmware.

    Allowed values: `BareMetalIPXEFirmware`
  - **settings.nodes.parameters.ironic.ipxeFirmwareRef.name**
    string

Required value

Name of the BareMetalIPXEFirmware resource.

Length: `1..253`

- **settings.nodes.parameters.ironic.provisioningNetwork**
  object

  Settings of the provisioning network where servers boot over PXE.

  Used only if the module deploys Ironic in the cluster.

  - **settings.nodes.parameters.ironic.provisioningNetwork.interface**
    string

    Required value

    Name of the network interface connected to the provisioning network on the node where Ironic runs.

    Pattern: `^[A-Za-z0-9_.:-]{1,15}$`

  - **settings.nodes.parameters.ironic.provisioningNetwork.ipAddress**
    string

    Required value

    Free IPv4 address in the provisioning network.

    Ironic assigns this address to the interface from the `interface` parameter and accepts server connections on it.

- **settings.nodes.parameters.ironic.provisioningTLS**
  object

  Enables TLS for the Ironic API and for file downloads by servers.

  Supported only with the built-in DHCP server (`dhcp.internal`). Requires the `ramdiskImageRef` and `ipxeFirmwareRef` parameters. The module creates Ironic only when the Secret and the resources from these parameters are ready.

  - **settings.nodes.parameters.ironic.provisioningTLS.certificateRef**
    object

    Required value

    Reference to the Secret with the Ironic TLS certificate in the `d8-infra-provider-baremetal` namespace.

    The Secret must have the `kubernetes.io/tls` type, contain the `tls.crt`, `tls.key`, and `ca.crt` keys, and have the `environment.metal3.io/ironic-standalone-operator: "true"` label.

    - **settings.nodes.parameters.ironic.provisioningTLS.certificateRef.name**
      string

      Required value

      Name of the Secret with the Ironic TLS certificate.

      Length: `1..253`

- **settings.nodes.parameters.ironic.ramdiskImageRef**
  object

  Reference to the BareMetalRamdiskImage resource with the Ironic Python Agent (IPA) image for server boot.

  If the parameter is not set, the built-in IPA image is used. The parameter is required if the `provisioningTLS` parameter is set.

  - **settings.nodes.parameters.ironic.ramdiskImageRef.kind**
    string

    Required value

    Kind of the resource with the IPA image.

    Allowed values: `BareMetalRamdiskImage`

  - **settings.nodes.parameters.ironic.ramdiskImageRef.name**
    string

    Required value

    Name of the BareMetalRamdiskImage resource.

    Length: `1..253`

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom resources

i The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

⚠ The module lifecycle stage: Experimental

The module has requirements for installation

Before enabling the module, you need to allow the use of experimental modules in the `deckhouse` ModuleConfig. To do this, set the allowExperimentalModules parameter to `true`.

BareMetalImage

Scope: Cluster
Version: v1

- **metadata**
  object
- **spec**
  object

  Required value

  Operating system disk image installed on physical servers.

  - **spec.direct**
    object

    Required value

Image source and data for its integrity check.

Cannot be changed after creation. To update the operating system image, create a BareMetalImage with a new name and update the imageRef in BareMetalInstanceClass.

- **spec.direct.checksum**
  string

  Required value

  Image checksum in hexadecimal format.

  Its length must match the algorithm in the `checksumType` parameter.

  Pattern: `^[a-fA-F0-9]+$`

  Length: `1..128`

- **spec.direct.checksumType**
  string

  Required value

  Checksum algorithm.

  Allowed values: `md5`, `sha256`, `sha512`

- **spec.direct.format**
  string

  Required value

  Disk image format.

  Allowed values: `qcow2`, `raw`

- **spec.direct.url**
  string

  Required value

  Image URL.

  The image must be available at this URL to Ironic and to servers during operating system installation.

  Pattern: `^https?://`

  Maximum length: `2048`

BareMetalInstance

Short names: bmi

Scope: Cluster
Version: v1

- **metadata**
  object
- **spec**
  object

  Required value

  Cluster-wide physical server and settings for access to its baseboard management controller (BMC).

  - **spec.bmc**
    object

    Required value

    Settings for connecting to the server BMC.

    - **spec.bmc.allowHTTP**
      boolean

      Allows connecting to the BMC using Redfish over HTTP if the HTTPS connection fails.

      In this case, BMC credentials are sent in plain text. Cannot be used with the `tls.caCertRef` parameter.

      Default: `false`

    - **spec.bmc.credentialsRef**
      object

      Required value

      Reference to the Secret with BMC credentials. If namespace is omitted, the Secret is read from `d8-infra-provider-baremetal`.

      The Secret must have the `cloud-provider.deckhouse.io/credentials` type and contain the following keys:

      - `authScheme` set to `userPassword`
      - `identity` with the BMC user name
      - `secret` with the BMC user password
      - **spec.bmc.credentialsRef.kind**
        string

        Required value

        Kind of the resource with BMC credentials.

        Allowed values: `Secret`

      - **spec.bmc.credentialsRef.name**
        string

        Required value

        Name of the Secret with BMC credentials.

        Length: `1..253`

      - **spec.bmc.credentialsRef.namespace**
        string

        Namespace of the Secret with BMC credentials.

        Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

        Length: `1..63`

    - **spec.bmc.insecure**
      boolean

      Disables BMC certificate verification.

      Cannot be used with the `tls.caCertRef` parameter.

Default: `false`

- **spec.bmc.ipAddress**
  string

  Required value

  BMC IPv4 address.

- **spec.bmc.port**
  integer

  BMC port.

  If the parameter is set, the module connects to the BMC on this port over all protocols. If the parameter is not set, the module uses the standard ports:

  - `443` for Redfish over HTTPS
  - `80` for Redfish over HTTP if the `allowHTTP` parameter is enabled
  - `623` for IPMI

  Allowed values: `1 <= X <= 65535`

- **spec.bmc.systemUUID**
  string

  Required value

  Server UUID reported by its BMC.

  Over Redfish, the module compares this value with the system UUID returned by the BMC, and over IPMI with the system identifier (System GUID).

- **spec.bmc.tls**
  object

  Settings for verifying the BMC certificate when connecting over Redfish.

  - **spec.bmc.tls.caCertRef**
    object

    Reference to the Secret key with the PEM-encoded CA certificate that issued the BMC certificate.

    The Secret must be in the `d8-infra-provider-baremetal` namespace.

    Supported only if Ironic runs in the cluster. If the module is connected to an external Ironic, an error appears in the resource status, and the module does not create a BareMetalHost for it.

    - **spec.bmc.tls.caCertRef.key**
      string

      Required value

      Key in the Secret that holds the PEM-encoded BMC CA certificate.

      Length: `1..253`

    - **spec.bmc.tls.caCertRef.kind**
      string

      Required value

      Kind of the resource with the BMC CA certificate.

      Allowed values: `Secret`

    - **spec.bmc.tls.caCertRef.name**
      string

      Required value

      Name of the Secret with the BMC CA certificate.

      Length: `1..253`

- **spec.bootMACAddress**
  string

  Required value

  MAC address of the server network interface used for PXE boot.

  Pattern: `^([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}$`

- **spec.online**
  boolean

  Desired power state of the server.

  Applied when the physical host is created and when this parameter changes on a host that is not used for a node.

  While the server is used for a node, CAPM3 controls its power; changes to this parameter are ignored. After node deletion, the module preserves the power state requested by CAPM3 until this parameter is changed explicitly.

  Default: `false`

- **spec.rootDeviceHints**
  object

  Root disk hints.

  If the parameter is not set, Ironic selects the root disk by itself based on the hardware information.

  - **spec.rootDeviceHints.deviceName**
    string

    Linux disk name, for example, `/dev/vda`, or a path to the disk in the `/dev/disk/by-path/` directory.

    Only a disk with exactly the same value matches.

    Maximum length: `256`

  - **spec.rootDeviceHints.hctl**
    string

    Disk address on the SCSI bus, for example, `0:0:0:0`.

    Only a disk with exactly the same address matches.

    Maximum length: `256`

  - **spec.rootDeviceHints.minSizeGigabytes**
    integer

    Minimum disk size in gigabytes.

    Allowed values: `0 <= X`

  - **spec.rootDeviceHints.model**
    string

Disk model.

A disk matches if its model name contains the specified string. For example, `860 EVO` matches the `Samsung SSD 860 EVO 500GB` model.

Maximum length: `256`

- **spec.rootDeviceHints.rotational**
  boolean

  Disk type.

  If `true`, a hard disk drive (HDD) is selected. If `false`, a solid-state drive, for example, SSD or NVMe, is selected.

- **spec.rootDeviceHints.serialNumber**
  string

  Disk serial number.

  Only a disk with exactly the same serial number matches.

  Maximum length: `256`

- **spec.rootDeviceHints.vendor**
  string

  Disk vendor.

  A disk matches if its vendor name contains the specified string. For example, `Samsung` matches the `Samsung Electronics` vendor.

  Maximum length: `256`

- **spec.rootDeviceHints.wwn**
  string

  World Wide Name (WWN) of the disk.

  Only a disk with exactly the same WWN matches.

  Maximum length: `256`

- **spec.rootDeviceHints.wwnVendorExtension**
  string

  Vendor extension of the disk WWN.

  Only a disk with exactly the same value matches.

  Maximum length: `256`

- **spec.rootDeviceHints.wwnWithExtension**
  string

  Disk WWN together with the vendor extension.

  Only a disk with exactly the same value matches.

  Maximum length: `256`

- **status**
  object

  Current server state and the result of processing the resource by the module.

  - **status.conditions**
    array of objects

    Conditions that describe the resource state.

    Each time the module processes the resource, it sets the `BMCResolved` condition. The condition is `True` if the module connected to the BMC, found the server with the UUID from the `spec.bmc.systemUUID` parameter, and created or updated the BareMetalHost resource for it.

    If the module failed to complete one of these steps, the condition is `False`. The `reason` field identifies the step, and the `message` field explains the error.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition value or reason last changed.

    - **status.conditions.message**
      string

      Required value

      Explanation of the condition value.

      The message is empty if the condition is `True`. The main messages for the `False` value are described in the Checking server state section.

    - **status.conditions.observedGeneration**
      integer

      Resource generation (`metadata.generation`) for which the condition was set.

    - **status.conditions.reason**
      string

      Required value

      Code of the reason why the condition has its current value.

      If the condition is `True`, the reason is `BMCResolved`. If the condition is `False`, the reason identifies the step that the module failed to complete, for example `CredentialsInvalid` or `BMCResolutionFailed`.

    - **status.conditions.status**
      string

      Required value

      Condition value.

      The value is `True` or `False`.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      Condition type.

  - **status.host**
    object

    Server state according to the BareMetalHost resource with the same name in the `d8-cloud-instance-manager` namespace.

    The field is filled in only if the last processing of the resource succeeded.

        - **status.host.desiredOnline**
          boolean

          Desired power state of the server.

        - **status.host.error**
          string

          Error message if an operation on the server failed.

          The error details are in the `status.errorMessage` field of the BareMetalHost resource with the same name.

        - **status.host.poweredOn**
          boolean

          Actual power state of the server.

        - **status.host.state**
          string

          Current server state according to the BareMetalHost resource, for example, `available` or `provisioned`.

    - **status.message**
      string

      Error message if the module failed to process the resource.

    - **status.observedGeneration**
      integer

      Resource generation (`metadata.generation`) last processed by the module.

BareMetalInstanceClass

Scope: Cluster
Version: v1

- **metadata**
  object
- **spec**
  object

  Required value

  Parameters of CloudEphemeral nodes created on physical servers.

    - **spec.capacity**
      object

      Node resources that cluster autoscaler takes into account when scaling the group from zero.

        - **spec.capacity.cpu**
          integer or string

          Required value

          Node CPU resources available for workloads.

          Pattern: `^[0-9]+m?$`

          Example:

          ```
          cpu: 4000m
          ```

        - **spec.capacity.memory**
          integer or string

          Required value

          Node memory available for workloads.

          Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

          Example:

          ```
          memory: 8Gi
          ```

    - **spec.disableAutomatedCleaning**
      boolean

      Disables automatic disk cleaning on servers used for the group nodes.

      By default, Ironic cleans disk metadata.

      Default: `false`

    - **spec.hostSelector**
      object

      Selector that picks servers by the labels of BareMetalInstance resources.

      If the parameter is not set, nodes are created on any free servers.

        - **spec.hostSelector.matchExpressions**
          array of objects

          Selects servers using Kubernetes label selector expressions.

            - **spec.hostSelector.matchExpressions.key**
              string

              Required value

              Label key.

              Maximum length: `253`

            - **spec.hostSelector.matchExpressions.operator**
              string

              Required value

              Operator that compares the label value with the values from the `values` parameter.

              Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

            - **spec.hostSelector.matchExpressions.values**
              array of strings

              Label values for the `In` and `NotIn` operators.

              Do not set for the `Exists` and `DoesNotExist` operators.

                - **Element of the array**

string

Maximum length: `63`

- **spec.hostSelector.matchLabels**
  object

  Selects servers that have all the listed labels with the specified values.

- **spec.imageRef**
  object

  Required value

  Reference to the BareMetalImage resource with the operating system image for the nodes.

  - **spec.imageRef.kind**
    string

    Required value

    Kind of the resource with the operating system image.

    Allowed values: `BareMetalImage`

  - **spec.imageRef.name**
    string

    Required value

    Name of the BareMetalImage resource.

    Length: `1..253`

- **spec.nodeReuse**
  boolean

  Allows reusing the same servers when the group nodes are updated.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

    NodeGroups that use this instance class.

BareMetalIPXEFirmware

Scope: Cluster
Version: v1

- **metadata**
  object
- **spec**
  object

  Required value

  iPXE firmware for booting servers if TLS is enabled for Ironic in the cluster.

  - **spec.direct**
    object

    Required value

    Firmware files and their SHA-256 checksums for the supported boot modes.

    - **spec.direct.bios**
      object

      Required value

      iPXE firmware for legacy BIOS boot.

      - **spec.direct.bios.sha256**
        string

        Required value

        SHA-256 checksum of the iPXE firmware for BIOS.

        Pattern: `^[a-f0-9]{64}$`

      - **spec.direct.bios.url**
        string

        Required value

        URL of the iPXE firmware for BIOS.

        Only HTTPS is supported.

        Pattern: `^https://`

        Maximum length: `2048`

    - **spec.direct.uefiArm64**
      object

      iPXE firmware for UEFI boot on ARM64.

      If the parameter is not set, the built-in firmware is used.

      - **spec.direct.uefiArm64.sha256**
        string

        Required value

        SHA-256 checksum of the iPXE firmware for UEFI on ARM64.

        Pattern: `^[a-f0-9]{64}$`

      - **spec.direct.uefiArm64.url**
        string

        Required value

        URL of the iPXE firmware for UEFI on ARM64.

        Only HTTPS is supported.

        Pattern: `^https://`

        Maximum length: `2048`

- **spec.direct.uefiX86_64**
  object

  Required value

  iPXE firmware for UEFI boot on x86_64.

  - **spec.direct.uefiX86_64.sha256**
    string

    Required value

    SHA-256 checksum of the iPXE firmware for UEFI on x86_64.

    Pattern: `^[a-f0-9]{64}$`

  - **spec.direct.uefiX86_64.url**
    string

    Required value

    URL of the iPXE firmware for UEFI on x86_64.

    Only HTTPS is supported.

    Pattern: `^https://`

    Maximum length: `2048`

BareMetalRamdiskImage

Scope: Cluster
Version: v1

- **metadata**
  object
- **spec**
  object

  Required value

  Ironic Python Agent (IPA) image that servers boot from while Ironic collects hardware information and installs the operating system.

  - **spec.direct**
    object

    Required value

    URLs of the IPA kernel and initramfs.

    - **spec.direct.architecture**
      string

      CPU architecture the image is built for.

      Default: `x86_64`

      Allowed values: `x86_64`, `aarch64`

    - **spec.direct.initramfsURL**
      string

      Required value

      URL of the IPA initramfs.

      The file must be available at this URL to Ironic and to servers.

      Pattern: `^https?://`

      Maximum length: `2048`

    - **spec.direct.kernelURL**
      string

      Required value

      URL of the IPA kernel.

      The file must be available at this URL to Ironic and to servers.

      Pattern: `^https?://`

      Maximum length: `2048`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> The module lifecycle stage: Experimental
>
> The module has requirements for installation
>
> Before enabling the module, you need to allow the use of experimental modules in the `deckhouse` ModuleConfig. To do this, set the allowExperimentalModules parameter to `true`.

Choosing the Ironic mode

Ironic manages server power through their baseboard management controllers (BMCs) and installs an operating system on servers. The module can deploy Ironic in the cluster by itself or connect to Ironic deployed outside the cluster. The Ironic mode is set in the `nodes.parameters.ironic` parameter.

To deploy Ironic in the cluster, set the `provisioningNetwork` and `dhcp` parameters. To connect to an external Ironic, set the `externalInstance` parameter. Parameters of different modes cannot be set at the same time.

Ironic in the cluster

Ironic uses the network interfaces of the node it runs on. The module does not pin Ironic to a specific node, so every node where Ironic can start must be connected to the provisioning network where servers boot over PXE. Specify the name of the network interface connected to this network in the `provisioningNetwork.interface` parameter.

In the `provisioningNetwork.ipAddress` parameter, specify a free IPv4 address in the provisioning network. Ironic assigns this address to the interface by itself and accepts connections on it on the following TCP ports:

Port	Purpose
`6180`	HTTP server with images for server boot
`6385`	Ironic API
`6183`	Virtual media HTTPS server if TLS is enabled
`8084`	HTTPS server for iPXE if TLS is enabled

For PXE boot, servers get network settings from a DHCP server. To have Ironic assign addresses to servers by itself, set the `dhcp.internal` parameter with the subnet and the address range. If a DHCP server already runs in the provisioning network, set the `dhcp.external` parameter and configure PXE boot on that DHCP server yourself.

During preparation, servers boot from the Ironic Python Agent (IPA) service image. By default, the built-in IPA image is used, referenced by the `baremetal-default-ramdisk` BareMetalRamdiskImage resource. The module creates this resource by itself. To use a different image, create your own BareMetalRamdiskImage resource and set its name in the `ramdiskImageRef` parameter.

External Ironic

In this mode, the module does not deploy Ironic or a DHCP server in the cluster, and network boot of servers is provided by the external Ironic. Set its API address in the `externalInstance.endpoint` parameter. The address must end with `/v1` or `/v1/`.

If the API requires authentication, create a Secret with credentials in the `d8-infra-provider-baremetal` namespace and specify it in the `externalInstance.credentialsRef` parameter.

To have the Ironic API certificate verified on connection, specify the CA certificate in the `externalInstance.tls.caCertRef` parameter. If Ironic requires a client certificate, specify it in the `externalInstance.tls.clientCertRef` parameter.

The module does not pass BMC CA certificates to the external Ironic, and for BareMetalInstance resources with the `spec.bmc.tls.caCertRef` parameter it reports an error and does not create a BareMetalHost. Configure trust for BMC certificates in the external Ironic itself.

Initializing Kubernetes nodes

The module deploys its Cloud Controller Manager for both managed and external Ironic. It matches each Node to its CAPI Machine, Metal3Machine, and BareMetalHost, sets the Metal3 providerID, and removes `node.cloudprovider.kubernetes.io/uninitialized` after initialization. It does not remove the bashible initialization taint.

A module bashible step publishes the installed operating system's internal IP address for CCM. On first boot, it uses the source address of the route to the control plane. Addresses collected by IPA during inspection are not used as node addresses, since the provisioning network can differ from the node network. OS images must support cloud-init and apply the hostname supplied by Metal3DataTemplate.

Registering physical servers

To make a server available for nodes, describe it in a BareMetalInstance resource. BareMetalInstance is cluster-scoped: omit `metadata.namespace` from its manifest. Specify the namespace of the BMC credentials Secret in `spec.bmc.credentialsRef.namespace`. If omitted, `d8-infra-provider-baremetal` is used.

The requirements for the credentials Secret are listed in the description of the `spec.bmc.credentialsRef` parameter.

Connecting to the BMC

The BMC address is set in the `spec.bmc.ipAddress` parameter. The module connects to the BMC over the Redfish or IPMI protocol. It checks the protocols in the following order and uses the first one through which the server is found:

1. Redfish over HTTPS
2. Redfish over HTTP if the `spec.bmc.allowHTTP` parameter is set to `true`
3. IPMI

By default, the module uses port `443` for Redfish over HTTPS, port `80` for Redfish over HTTP, and port `623` for IPMI. If the BMC accepts connections on a different port, specify it in the `spec.bmc.port` parameter. In this case, the module uses the specified port for all protocols. The timeout of each check is set by the `bmcProbeTimeoutSeconds` parameter in the module settings.

In the `spec.bmc.systemUUID` parameter, specify the server UUID reported by its BMC. The module uses this UUID to select the server among all servers managed by the BMC. If there is no server with this UUID, the module reports an error and does not register the server.

The module verifies the BMC certificate against the system root certificates. If the BMC certificate is issued by a private CA, create a Secret with the PEM-encoded CA certificate in the `d8-infra-provider-baremetal` namespace and specify it in the `spec.bmc.tls.caCertRef` parameter. Ironic in the cluster uses a common set of CA certificates from all BareMetalInstance resources, so a CA certificate specified for one server is trusted when connecting to the BMC of any server.

Server parameters

In the `spec.bootMACAddress` parameter, specify the MAC address of the network interface the server boots from over PXE. If the server has several disks, specify the root disk hints in the `spec.rootDeviceHints` parameter.

Add labels to the BareMetalInstance resource so that an instance class can select the server for nodes. The selection rules are set in the `hostSelector` parameter of the instance class.

Checking server state

To check the server state, run the following command:

```
d8 k get baremetalinstances
```

Example output:

```
NAME        BMC              BOOT MAC            STATE       DESIRED ONLINE   POWERED ON   ERROR   AGE
worker-01   192.168.201.32   f0:3f:03:01:1d:22   available   false            false                15m
```

The server is ready for a node when the `STATE` column shows `available`. If the module fails to connect to the BMC or to prepare resources for the server, the `ERROR` column shows an error message. The main error messages:

Message	Description
`BMC credentials are invalid or unavailable.`	The credentials Secret is not found, has a different type, or lacks the required keys
`Unable to resolve the BMC endpoint.`	The BMC is unreachable over all protocols, rejects the credentials, or does not report a server with the specified `systemUUID`
`Unable to synchronize the BMC CA certificate.`	The CA certificate Secret is not found or does not contain a PEM-encoded CA certificate, or Ironic in the cluster has not been created yet
`BMC CA certificates are not supported with an external Ironic instance.`	The `spec.bmc.tls.caCertRef` parameter is set while connected to an external Ironic
`Unable to synchronize the BMC credentials Secret.`	Failed to create or update the service Secret with BMC credentials
`Unable to synchronize the physical host.`	Failed to create or update the BareMetalHost resource for the server
`A generated resource with the required name is owned by another object.`	The BareMetalHost resource or the service Secret for this server already exists and belongs to another BareMetalInstance resource or was created manually

Creating nodes

The `node-manager` module creates nodes on physical servers in node groups with the `CloudEphemeral` type. Such groups need a BareMetalImage resource with the operating system image and a BareMetalInstanceClass resource with node parameters.

Ironic installs the image from the BareMetalImage resource on the server, so the image must meet the module requirements. The instance class refers to the image and selects servers by the labels of BareMetalInstance resources.

The `spec.direct` parameters of BareMetalImage are immutable. To update the operating system image, create a BareMetalImage with a new name and change `imageRef` in the instance class. This change starts a rollout for groups using that instance class.

If the node group can scale from zero (the `minPerZone` parameter is `0`), specify the server CPU and memory in the `capacity` parameter of the instance class. Cluster autoscaler uses these values while the group has no nodes. Without them, the `Capacity calculation failed` error appears in the node group status.

To create nodes, create a node group (NodeGroup) with the `CloudEphemeral` type and specify the instance class in the `cloudInstances.classReference` parameter. Physical servers use a single `default` zone, so you can omit the `cloudInstances.zones` parameter.

Nodes of the group are recreated if you change one of the following parameters:

- The image reference in the `imageRef` parameter
- The `hostSelector` parameter
- The `disableAutomatedCleaning` parameter
- The `nodeReuse` parameter

Enabling TLS for server boot

By default, servers download images and access Ironic over HTTP. You can protect this traffic with TLS only in the Ironic in the cluster mode with the built-in DHCP server (the `dhcp.internal` parameter). If you set the TLS parameters together with the `dhcp.external` parameter, the module does not create Ironic.

Prepare the following resources for TLS:

- A Secret with the Ironic certificate. The Secret requirements are listed in the description of the `provisioningTLS.certificateRef` parameter. The certificate must be issued for the address from the `provisioningNetwork.ipAddress` parameter.
- A BareMetalRamdiskImage resource with the IPA image.
- A BareMetalIPXEFirmware resource with iPXE firmware for BIOS and UEFI x86_64. The firmware must trust the CA from the `ca.crt` key of the Secret, because iPXE downloads files from Ironic over HTTPS. The firmware itself must be available over HTTPS on a server whose certificate is issued by the same CA.

Specify these resources in the `provisioningTLS.certificateRef`, `ramdiskImageRef`, and `ipxeFirmwareRef` parameters. The module creates Ironic only when all three resources exist and the Secret meets the requirements.

⚠ Do not delete the resources specified in the TLS parameters while these parameters are set. If any of the resources becomes unavailable, the module deletes Ironic, and server boot stops.

An example of TLS setup is provided on the “Examples” page.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

⚠ The module lifecycle stage: Experimental

The module has requirements for installation

Before enabling the module, you need to allow the use of experimental modules in the `deckhouse` ModuleConfig. To do this, set the allowExperimentalModules parameter to `true`.

Creating a node with Ironic in the cluster

In this example, Ironic runs in the cluster with the built-in DHCP server, and one CloudEphemeral node is created on a physical server. Replace the addresses, credentials, server UUID, and image data with values of your environment.

1. Enable the module and configure Ironic:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: infra-provider-baremetal
spec:
  enabled: true
  version: 1
  settings:
    nodes:
      parameters:
        ironic:
          provisioningNetwork:
            interface: enp7s0
            ipAddress: 192.168.223.85
          dhcp:
            internal:
              networkCIDR: 192.168.223.0/24
              rangeBegin: 192.168.223.160
              rangeEnd: 192.168.223.171
              gatewayAddress: 192.168.223.254
              dnsAddress: 192.168.223.1
```

2. Create a Secret with credentials of the server baseboard management controller (BMC) and a BareMetalInstance resource that describes the server:

```
apiVersion: v1
kind: Secret
metadata:
  name: worker-01-bmc
  namespace: d8-infra-provider-baremetal
type: cloud-provider.deckhouse.io/credentials
stringData:
  authScheme: userPassword
  identity: <BMC_USER>
  secret: <BMC_PASSWORD>
---
apiVersion: deckhouse.io/v1
kind: BareMetalInstance
metadata:
  name: worker-01
  labels:
    pool: workers
spec:
  bootMACAddress: "f0:3f:03:01:1d:22"
  bmc:
    ipAddress: 192.168.201.32
    systemUUID: "00583f1d-2903-e203-0010-d21de0c7571b"
    credentialsRef:
      namespace: d8-infra-provider-baremetal
      kind: Secret
      name: worker-01-bmc
```

Replace `<BMC_USER>` and `<BMC_PASSWORD>` with the BMC user name and password. The instance class from the fourth step selects the server by the `pool: workers` label.

3. Wait until the server moves to the `available` state:

```
d8 k get baremetalinstances worker-01
```

4. Create a BareMetalImage resource with the operating system image and a BareMetalInstanceClass resource with node parameters:

```
apiVersion: deckhouse.io/v1
kind: BareMetalImage
metadata:
  name: ubuntu-24-04-amd64
spec:
  direct:
    url: http://images.example.com/ubuntu-24.04-amd64.qcow2
    checksum: 6695ad25a7a930accfd1e004f151578c4152338191eb7598fd89745ab93d3160
    checksumType: sha256
    format: qcow2
---
apiVersion: deckhouse.io/v1
kind: BareMetalInstanceClass
metadata:
  name: workers
spec:
  imageRef:
    kind: BareMetalImage
    name: ubuntu-24-04-amd64
  hostSelector:
    matchLabels:
      pool: workers
  capacity:
    cpu: "4"
    memory: 8Gi
```

5. Create a NodeGroup with the `CloudEphemeral` node type:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1
    maxPerZone: 1
    classReference:
      kind: BareMetalInstanceClass
      name: workers
```

6. Check that the node has appeared in the cluster:

```
d8 k get nodes -l node.deckhouse.io/group=workers
```

Once Ironic installs the operating system and the server joins the cluster, the node appears in the command output.

Connecting to an external Ironic

In this example, the module connects to Ironic deployed outside the cluster, with password authentication and API certificate verification.

1. Create a Secret with Ironic API credentials and a Secret with the CA certificate:

```
d8 k -n d8-infra-provider-baremetal create secret generic ironic-credentials \
  --from-literal=identity=<IRONIC_USER> \
  --from-literal=secret=<IRONIC_PASSWORD>
d8 k -n d8-infra-provider-baremetal create secret generic ironic-ca \
  --from-file=ca.crt=<CA_FILE>
```

Replace `<IRONIC_USER>` and `<IRONIC_PASSWORD>` with the Ironic API user name and password, and `<CA_FILE>` with the path to the file with the CA certificate that signed the API certificate.

2. Enable the module and set the connection settings in the `externalInstance` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: infra-provider-baremetal
spec:
  enabled: true
  version: 1
  settings:
    nodes:
      parameters:
        ironic:
          externalInstance:
            endpoint: https://ironic.example.com:6385/v1/
            credentialsRef:
              kind: Secret
              name: ironic-credentials
              namespace: d8-infra-provider-baremetal
            tls:
              caCertRef:
                kind: Secret
                name: ironic-ca
                namespace: d8-infra-provider-baremetal
                key: ca.crt
```

Servers and nodes in this mode are created in the same way as in the previous example, starting from the second step.

Enabling TLS for server boot

In this example, the module protects server boot traffic with TLS. The example extends the settings from the first example.

1. Create a Secret with the Ironic certificate and add a label to it:

   ```
   d8 k -n d8-infra-provider-baremetal create secret generic ironic-tls \
     --type=kubernetes.io/tls \
     --from-file=tls.crt=<CERT_FILE> \
     --from-file=tls.key=<KEY_FILE> \
     --from-file=ca.crt=<CA_FILE>
   d8 k -n d8-infra-provider-baremetal label secret ironic-tls \
     environment.metal3.io/ironic-standalone-operator=true
   ```

   Replace `<CERT_FILE>` and `<KEY_FILE>` with the paths to the Ironic certificate and its private key, and `<CA_FILE>` with the path to the CA certificate that issued the Ironic certificate.

2. Create a BareMetalRamdiskImage resource with the Ironic Python Agent (IPA) image and a BareMetalIPXEFirmware resource with iPXE firmware:

   ```
   apiVersion: deckhouse.io/v1
   kind: BareMetalRamdiskImage
   metadata:
     name: ipa-tls
   spec:
     direct:
       kernelURL: https://images.example.com/ipa/ironic-python-agent.kernel
       initramfsURL: https://images.example.com/ipa/ironic-python-agent.initramfs
   ---
   apiVersion: deckhouse.io/v1
   kind: BareMetalIPXEFirmware
   metadata:
     name: ipxe-tls
   spec:
     direct:
       bios:
         url: https://images.example.com/ipxe/undionly.kpxe
         sha256: <BIOS_FIRMWARE_SHA256>
       uefiX86_64:
         url: https://images.example.com/ipxe/snponly.efi
         sha256: <UEFI_FIRMWARE_SHA256>
   ```

   Replace `<BIOS_FIRMWARE_SHA256>` and `<UEFI_FIRMWARE_SHA256>` with the lowercase SHA-256 checksums of the firmware files.

3. Add the `provisioningTLS`, `ramdiskImageRef`, and `ipxeFirmwareRef` parameters to the module settings:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: infra-provider-baremetal
   spec:
     enabled: true
     version: 1
     settings:
       nodes:
         parameters:
           ironic:
             provisioningNetwork:
               interface: enp7s0
               ipAddress: 192.168.223.85
             dhcp:
               internal:
                 networkCIDR: 192.168.223.0/24
                 rangeBegin: 192.168.223.160
                 rangeEnd: 192.168.223.171
             provisioningTLS:
               certificateRef:
                 name: ironic-tls
             ramdiskImageRef:
               kind: BareMetalRamdiskImage
               name: ipa-tls
             ipxeFirmwareRef:
               kind: BareMetalIPXEFirmware
               name: ipxe-tls
   ```

4. Check that the module has created Ironic:

   ```
   d8 k -n d8-infra-provider-baremetal get ironics.ironic.metal3.io
   ```

   If the output has no `ironic` resource, check that all three resources from the TLS parameters are created and the Secret meets the requirements.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The istio module

> Available with limitations in: **Open/CE, Core, BE, SE, SE+, Certified Core/CSE Lite (1.73)**
>
> Available without limitations in: **Ultimate/EE**

> Included with limitations in extensions: **Advanced Networking, Advanced Infrastructure Security, Cluster Union, Multitenancy, Network Security**

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

> Web interfaces associated with the module: istio

Compatibility table for supported versions

The table below shows Istio versions and their support status in Deckhouse Platform (DP).

Istio version	Kubernetes versions supported by Istio	Status in DP
1.29	1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36	Supported
1.27	1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36	Supported
1.25	1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36	Deprecated and will be deleted

> ⚠ Istio 1.21 is no longer supported. Before upgrading to this DP release, migrate every global and additional Istio 1.21 revision to a supported version using a previous DP release that supports both versions. The DP upgrade is blocked while Istio 1.21 remains configured.

Problems Istio helps to solve

Istio is a framework for managing network traffic on a centralized basis that implements the Service Mesh approach.

Istio solves the tasks for applications:

- Using Mutual TLS.
- Access authorization between services.
- Request routing.
- Managing request balancing between service Endpoints.
- Improving Observability.
- Organizing a multi-datacenter cluster by joining clusters into a single Service Mesh (multicluster).
- Grouping isolated clusters into a federation with the ability to provide native (in the Service Mesh sense) access to selected services .
- Joining the mesh without per-pod sidecars using ambient mode.

Mutual TLS

Mutual TLS is the main method of mutual service authentication. It is based on the fact that all outgoing requests are verified using the server certificate, and all incoming requests are verified using the client certificate. After the verification is complete, the sidecar-proxy can identify the remote node and use these data for authorization or auxiliary purposes.

Each service gets its own identifier of the following format: `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>` where `TrustDomain` is the cluster domain in our case. You can assign your own ServiceAccount to each service or use the regular “default” one. The service ID can be used for authorization and other purposes. This is the identifier used as a name to validate against in TLS certificates.

You can redefine this settings at the Namespace level.

Authorization

The AuthorizationPolicy resource is responsible for managing authorization. Once this resource is created for the service, the following algorithm is used for determining the fate of the request:

- The request is denied if it falls under the DENY policy.
- The request is allowed if there are no ALLOW policies for the service.
- The request is allowed if it falls under the ALLOW policy.
- In all other cases, the request is denied.

In other words, if you explicitly deny something, then only this restrictive rule will work. On the other hand, if you explicitly allow something, only explicitly authorized requests would be allowed (however, restrictions will have precedence).

You can use the following arguments for defining authorization rules:

- Service IDs and wildcard expressions based on them (`mycluster.local/ns/myns/sa/myapp` or `mycluster.local/*`)
- Namespace
- IP ranges
- HTTP headers
- JWT tokens

Request routing

VirtualService is the main resource for routing control; it allows you to override the destination of an HTTP or TCP request. Routing decisions can be based on the following parameters:

- Host or other headers
- URI
- Method (GET, POST, etc.)
- Pod labels or the namespace of the request source
- dst-IP or dst-port for non-HTTP requests

Managing request balancing between service Endpoints

DestinationRule is the main resource for managing request balancing; it allows you to configure the details of requests leaving the Pods:

- Limits/timeouts for TCP
- Balancing algorithms between Endpoints
- Rules for detecting problems on the Endpoint side to take it out of balancing
- Encryption details

> ⚠ All customizable limits apply to each client Pod individually (on a per Pod basis)! Suppose you limited a service to one TCP connection. In this case, if you have three client Pods, the service will get three incoming connections.

Observability

Tracing

Istio makes it possible to collect application traces and inject trace headers if there are none. In doing so, however, you have to keep in mind the following:

- If a request initiates secondary requests for a service, they must inherit the trace headers by means of the application.
- You will need to install Jaeger to collect and display traces.

Grafana

The standard module bundle includes the following additional dashboards:

- Dashboard for evaluating the throughput and success of requests/responses between applications.
- Dashboard for evaluating control plane performance and load.
- **Istio Configuration Health** (EE) — periodic configuration analysis findings: issue counts by type/severity/namespace, trends over time, and the current issue list (filterable by revision and namespace). Requires `configAnalysis.enabled`.

Kiali

Kiali is a tool for visualizing your application's service tree. It allows you to quickly assess the situation in the network connectivity by visualizing the requests and their quantitative characteristics directly on the scheme.

Mesh metrics and access logs (Telemetry API)

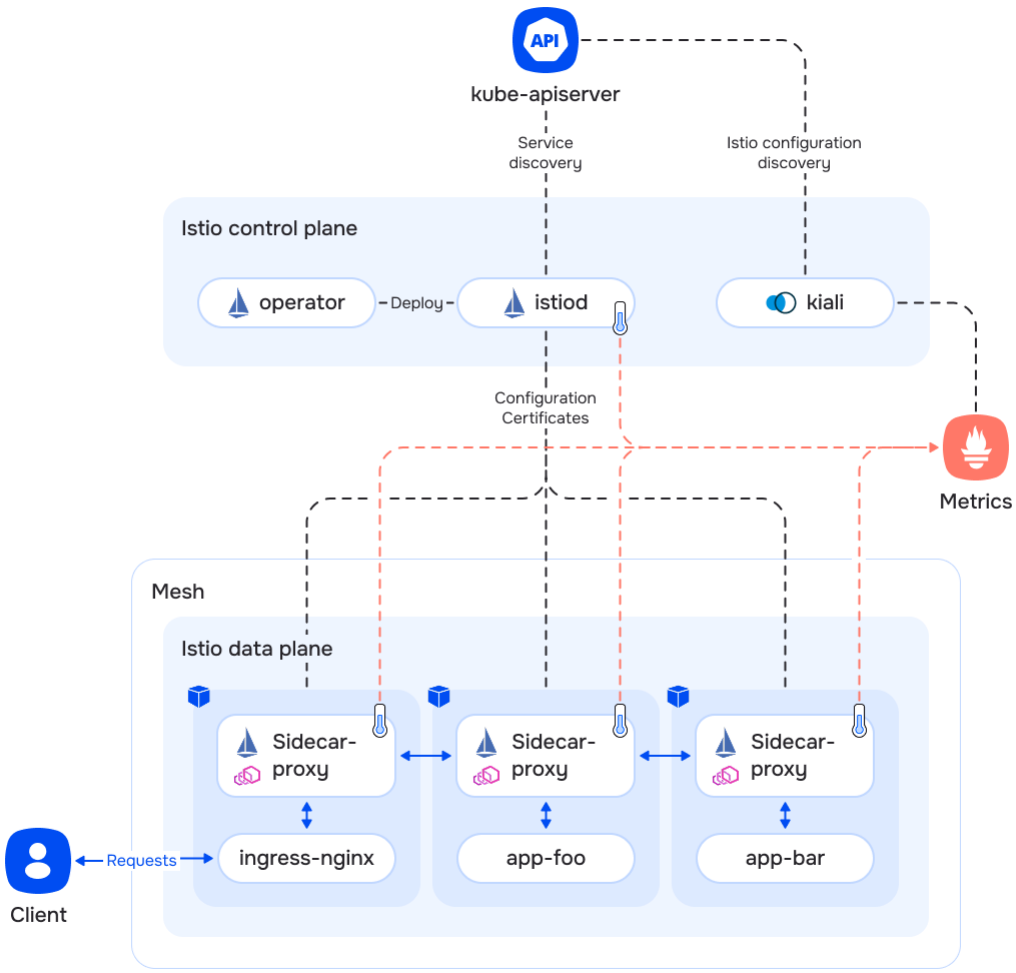
Metrics for Grafana workloads graphs and quantitative in Kiali on Prometheus scraping metrics `istio_*` series from `istio-proxy`. For the supported Istio versions, Istio exposes those using Telemetry API mesh defaults instead of legacy `telemetry.v2` filters alone.

DP configures `telemetryAPI.enabled`: when `false` you keep the legacy stack; when `true` you switch to `meshConfig.defaultProviders`, bundled `Telemetry` resources, and the `dataPlane.accessLog` template wired into access logs. Step-by-step examples, readiness checks, optional extra `Telemetry` policies, and tracing (`spec.tracing`) are in Telemetry API for mesh metrics, tracing, and access logs.

Architecture of the cluster with Istio enabled

The cluster components are divided into two categories:

- Control plane — managing and maintaining services; "control-plane" usually refers to istiod pods;
- Data plane — mediating and controlling all network communication between microservices, it is composed of a set of sidecar-proxy containers.



All data plane services are grouped into a mesh with the following features:

- It has a common namespace for generating service ID in the form `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. Each mesh has a TrustDomain ID (in our case, it is the same as the cluster domain), e.g. `mycluster.local/ns/myns/sa/myapp`.
- Services within a single mesh can authenticate each other using trusted root certificates.

Control plane components:

- `istiod` — the main service with the following tasks:
  - Continuous connection to the Kubernetes API and collecting information about services.
  - Processing and validating all Istio-related Custom Resources using the Kubernetes validating-webhook mechanism.
  - Configuring each sidecar proxy individually:
    - Generating authorization, routing, balancing rules, etc..
    - Distributing information about other application services in the cluster.
    - Issuing individual client certificates for implementing Mutual TLS. These certificates are unrelated to the certificates that Kubernetes uses for its own service needs.
  - Automatic tuning of manifests that describe application pods via the Kubernetes mutating-webhook mechanism:
    - Injecting an additional sidecar-proxy service container.
    - Injecting an additional init container for configuring the network subsystem (configuring DNAT to intercept application traffic).
    - Routing readiness and liveness probes through the sidecar-proxy.
- `operator` — installs all the resources required to operate a specific version of the control plane.
- `kiali` — dashboard for monitoring and controlling Istio resources as well as user services managed by Istio that allows you:
  - Visualize inter-service connections.
  - Diagnose problem inter-service connections.
  - Diagnose the control plane state.

The Ingress controller must be refined to receive user traffic:

- You need to add sidecar-proxy to the controller pods. It only handles traffic from the controller to the application services (the `enableIstioSidecar` parameter of the `IngressNginxController` resource).
- Services not managed by Istio continue to function as before, requests to them are not intercepted by the controller sidecar.
- Requests to services running under Istio are intercepted by the sidecar and processed according to Istio rules (read more about activating Istio to work with the application).

The istiod controller and sidecar-proxy containers export their own metrics that the cluster-wide Prometheus collects.

Application service architecture with Istio enabled

Details

- Each service pod gets a sidecar container — sidecar-proxy. From the technical standpoint, this container contains two applications:
  - Envoy proxies service traffic. It is responsible for implementing all the Istio functionality, including routing, authentication, authorization, etc.
  - pilot-agent is a part of Istio. It keeps the Envoy configurations up to date and has a built-in caching DNS server.
- Each pod has a DNAT configured for incoming and outgoing service requests to the sidecar-proxy. The additional init container is used for that. Thus, the traffic is routed transparently for applications.
- Since incoming service traffic is redirected to the sidecar-proxy, this also applies to the readiness/liveness traffic. The Kubernetes subsystem that does this doesn't know how to probe containers under Mutual TLS. Thus, all the existing probes are automatically reconfigured to use a dedicated sidecar-proxy port that routes traffic to the application unchanged.
- You have to configure the Ingress controller to receive requests from outside the cluster:
  - The controller's pods have additional sidecar-proxy containers.
  - Unlike application Pods, the Ingress controller's sidecar-proxy intercepts only outgoing traffic from the controller to the services. The incoming traffic from the users is handled directly by the controller itself;
- Ingress resources require refinement in the form of adding annotations:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — the Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is handled by the sidecar-proxy. Use this option only if your service has a ClusterIP.
  - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — the Ingress controller's sidecar-proxy makes routing decisions based on the Host header. If this annotation is omitted, the controller will leave a header with the site address (e.g. `Host: example.com`).
- Resources of the Service type do not require any adaptation and continue to function properly. Just like before, applications have access to service addresses like `servicename`, `servicename.myns.svc`, etc;
- DNS requests from within the pods are transparently redirected to the sidecar-proxy for processing:
  - This way, domain names of the services in the neighboring clusters can be disassociated from their addresses.

Native sidecars

By default, Istio 1.27 and newer inject sidecar-proxy as a Kubernetes native sidecar. The `istio-proxy` container is placed in `spec.initContainers` and has `restartPolicy: Always`. By default, Istio 1.25 injects a regular sidecar: `istio-proxy` is an ordinary container in `spec.containers`.

A native sidecar is tied to the Pod lifecycle:

- It starts before the application init containers, so they can use the Mesh network.
- It delays the start of the application containers until the proxy is ready.
- It stops only after the application containers have exited, so the application keeps network access while shutting down.
- It doesn't prevent Job Pods from completing.

Since the proxy is already running, the traffic of the application init containers goes through it and follows the same rules as the rest of the Pod's traffic: `outboundTrafficPolicyMode`, Sidecar egress settings, AuthorizationPolicy and mTLS. With a regular sidecar in the `InitContainer` traffic redirection mode, the application init containers run before the redirection is set up and bypass the mesh.

When a Pod moves from a regular sidecar to a native one, for example after an upgrade from Istio 1.25, check its init containers that connect to services outside the mesh. With `outboundTrafficPolicyMode: RegistryOnly`, register these services with a ServiceEntry.

To set the sidecar type for a specific Pod, add the `sidecar.istio.io/nativeSidecar` annotation (`"true"` or `"false"`) to the Pod template.

User request lifecycle

Application with Istio turned off

Application with Istio turned on

Activating Istio to work with the application

The main purpose of the activation is to add a sidecar container to the application pods so that Istio can manage the traffic.

The sidecar-injector is a recommended way to add sidecars. Istio can inject sidecar containers into user pods using the Admission Webhook mechanism. You can configure it using labels and annotations:

- A label attached to a namespace allows the sidecar-injector to identify a group of pods to inject sidecar containers into:
  - `istio-injection=enabled` — use the global version of Istio (`spec.settings.globalVersion` in `ModuleConfig`);
  - `istio.io/rev=v1x29` — use the specific Istio version for a given namespace;
  - `istio.io/rev=default` — use the global version of Istio (`spec.settings.globalVersion` in `ModuleConfig`).
- The `sidecar.istio.io/inject` (`"true"` or `"false"`) pod annotation lets you redefine the `sidecarInjectorPolicy` policy locally. These annotations work only in namespaces to which the above labels are attached.

It is also possible to add the sidecar to an individual pod in a namespace without the `istio-injection` and `istio.io/rev` labels by setting the `sidecar.istio.io/inject=true` Pod label. The same applies to the `istio.io/rev=vXxYZ` Pod label — it selects the Istio version for an individual pod, but only in a namespace that carries neither the `istio-injection` nor the `istio.io/rev` label.

The `istio-injection` and `istio.io/rev` namespace labels are mutually exclusive. A namespace carrying both of them, or carrying `istio-injection` with a value other than `enabled`, gets no sidecars injected at all — neither namespace-wide nor per-pod.

Istio-proxy, running as a sidecar container, consumes resources and adds the following overhead:

- Any incoming DNAT request is forcibly intercepted by the Envoy proxy. Envoy analyzes the request and forwards it by establishing a new connection. On the receiver side, the process is identical: traffic first reaches the "receiving" Envoy and only then is passed to the application itself.
- Each Envoy instance stores information about all services in the cluster; therefore, scaling the cluster leads to a linear increase in Envoy's RAM consumption due to storing the full service map. Using the Sidecar CustomResource, the configuration is filtered, and only the necessary minimum of data is delivered to Envoy.

> ⚠ The EnvoyFilter interface can be controlled by Lua plugins, but it is an internal control mechanism for implementing the Istio functionality. It must not be used in a user configuration, as doing so would compromise the integrity of the system.

It is also important to get the Ingress controller and the application's Ingress resources ready:

- Enable `enableIstioSidecar` of the `IngressNginxController` resource.
- Add annotations to the application's Ingress resources:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — the Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is now handled by the sidecar-proxy. Use this option only if your service has a ClusterIP.
  - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — the Ingress controller's sidecar-proxy makes routing decisions based on the Host header. If this annotation is omitted, the controller will leave a header with the site address (e.g. `Host: example.com`).

Federation and multicluster

DP supports two schemes of inter-cluster interaction:

- federation
- multicluster

Below are their fundamental differences:

- The federation aggregates multiple sovereign (independent) clusters:
  - Each cluster has its own namespace (for Namespace, Service, etc.);
  - Access to individual services between clusters is clearly defined.
- The multicluster aggregates co-dependent clusters:
  - Cluster namespaces are shared — each service is available to neighboring clusters as if it were running in a local cluster (unless authorization rules prohibit that).

Federation

Requirements for clusters

- Each cluster must have a unique domain set in the `clusterDomain` parameter of the `control-plane-manager` module. None of the clusters should use the default `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the `control-plane-manager` module may overlap.

Istio uses traffic analysis as follows:

- HTTP and HTTPS traffic — identification and routing or blocking decisions are based on headers.
- TCP traffic — decisions are based only on the destination IP address and port number.

⚠ `cluster.local` is an unmodified alias for the local cluster domain. Using `cluster.local` as a principal in `AuthorizationPolicy` always refers to the local cluster, even if another cluster in the mesh has `clusterDomain` explicitly set to `cluster.local` (Istio trust domain migration best practices).

i Istio operates in the multi-network mode: pods from different clusters can only communicate through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

⚠ Only sidecar-mode workloads can take part in a federation. For details, refer to Ambient mesh limitations.

General principles of federation

- Federation requires mutual trust between clusters. Thereby, to use federation, you have to make sure that both clusters (say, A and B) trust each other. This is achieved by a mutual exchange of root certificates.
- You also need to share information about public services to use the federation. You can do that using the `ServiceEntry` resource. A `ServiceEntry` defines the public `ingressgateway` address of the B cluster so that services of the A cluster can communicate with the bar service in the B cluster.

Enabling the federation

Enabling federation (via the `istio.federation.enabled = true` module parameter) results in the following activities:

- The `ingressgateway` service is added to the cluster. Its task is to proxy mTLS traffic coming from outside of the cluster to application services.
- A service gets added to the cluster that exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - List of public services in the cluster (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).

Managing the federation

To establish a federation, you must:

- Create a set of `IstioFederation` resources in each cluster that describe all the other clusters.
  - After successful auto-negotiation between clusters, the status of `IstioFederation` resource will be filled with neighbour's public and private metadata (`status.metadataCache.public` and `status.metadataCache.private`).
- Add the `federation.istio.deckhouse.io/public-service: ""` label to each Service that is considered public within the federation.
  - In the other federation clusters, corresponding `ServiceEntry` and DestinationRule resources will be created for each such Service, leading to the `ingressgateway` of the original cluster.
  - The label value must be empty. You do not need to label other resources, such as Deployment, Pod, or VirtualService, to publish a service in the federation.

⚠ Federation publishing does not support `ExternalName` services, services without `.spec.ports`, or services with ports missing the `name` field.

Each port name must start with a supported Istio prefix: `http`, `http2`, `https`, `tcp`, `tls`, `grpc`, or `grpc-web`. The module uses the port name to determine the protocol in the generated `ServiceEntry`. If the prefix is not recognized, the port will be handled as TCP.

Example of a public service:

```
apiVersion: v1
kind: Service
metadata:
  name: reviews
  namespace: bookinfo
  labels:
    federation.istio.deckhouse.io/public-service: ""
spec:
  selector:
    app: reviews
  ports:
  - name: http
    port: 9080
    targetPort: 9080
```

To troubleshoot service publishing in the federation, follow these checks:

Check which services are marked as public in the local cluster:

```
d8 k get svc -A -l federation.istio.deckhouse.io/public-service=
```

Check metadata exchange status with remote clusters:

```
d8 k get istiofederation
d8 k get istiofederation <name> -o jsonpath='{.status.conditions}'
```

Check that the remote cluster provided its public services list:

```
d8 k get istiofederation <name> -o jsonpath='{.status.metadataCache.private.publicServices}'
```

Check that local routing resources were created from the received metadata:

```
d8 k -n d8-istio get serviceentry,destinationrule
```

In the IstioFederation `status.conditions`, the `PublicMetadataExchangeReady`, `PrivateMetadataExchangeReady`, and `DataplaneConnectionReady` conditions should become `True`. If metadata exchange does not work, check the `D8IstioFederationMetadataEndpointDoesntWork` alert and the availability of the remote cluster `spec.metadataEndpoint`.

Multicluster

Requirements for clusters

- Cluster domains in the `clusterDomain` parameter of the `control-plane-manager` module must be the same for all multicluster members. The default value is `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the `control-plane-manager` module must be unique for each multicluster member.

Istio uses traffic analysis as follows:

- HTTP and HTTPS traffic — identification and routing or blocking decisions are based on headers.
- TCP traffic — decisions are based only on the destination IP address and port number.

> ⚠ If service or pod IP addresses overlap between clusters, requests from pods in other clusters may unintentionally match Istio's routing, allow, or deny rules. Overlapping service and pod subnets is not recommended (Istio network models).

By default, pods from different clusters communicate through the Istio ingress gateway (multi-network mode). If the pods of two clusters are directly reachable from each other (flat network), set `spec.enableIngressGateway` to `false` in both clusters, in the IstioMulticluster that describes the other cluster.

> ⚠ Only sidecar-mode workloads can take part in a multicluster. For details, refer to Ambient mesh limitations.

General principles

- Multicluster requires mutual trust between clusters. Thereby, to use multiclustering, you have to make sure that both clusters (say, A and B) trust each other. From a technical point of view, this is achieved by a mutual exchange of root certificates.
- Istio connects directly to the API server of the neighboring cluster to gather information about its services. This DP module takes care of the corresponding communication channel.

Enabling the multicluster

Enabling the multicluster (via the `istio.multicluster.enabled = true` module parameter) results in the following activities:

- A proxy is added to the cluster to publish access to the API server via the standard Ingress resource:
  - Access through this public address is secured by authorization based on Bearer tokens signed with trusted keys. DP automatically exchanges trusted public keys during the mutual configuration of the multicluster.
  - The proxy itself has read-only access to a limited set of resources.
- A service gets added to the cluster that exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - The public API server address (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).
  - Server public keys to authenticate requests to API server and to private metadata.

Managing the multicluster

To create a multicluster, you need to create a set of `IstioMulticluster` resources in each cluster that describe all the other clusters.

In case of issues when working with a multi-cluster, it is necessary to check in each cluster:

1. The conditions of the IstioMulticluster objects. To view them, run the following command:

   ```
   d8 k get istiomulticluster <NAME> -o jsonpath='{range .status.conditions[*]}{.type}: {.status}: {.reason}: {.message}{"\n"}{end}'
   ```

   Here, `<NAME>` is the name of the IstioMulticluster. Check every IstioMulticluster in the cluster. The `d8 k get istiomulticluster` command lists them.

   All conditions must be `True`. If a condition is not `True`, its `reason` and `message` say what is wrong.

2. Using the `istioctl` utility from the DP debug container, ensure that remote clusters have the `synced` status and a specified `istiod` instance (for details, refer to Debugging Istio with istioctl from the debug container):

   ```
   istioctl remote-clusters -i d8-istio
   ```

   Example output:

   ```
   NAME                        SECRET                                      STATUS     ISTIOD
   cluster-local-2451237693    d8-istio/istio-remote-secret-cluster-b      synced     istiod-v1x29-5c57d85b54-k8pl7
   ```

Ambient mesh

Besides the classic sidecar mode, Istio can run the data plane in *ambient* mode. In this mode, the mesh functionality is split into two layers, and application pods no longer get a per-pod `istio-proxy` sidecar container:

- A node-level component (`ztunnel`) handles L4 functionality (mutual TLS, identity, basic authorization) for all mesh workloads on the node.
- An optional per-namespace component (a *waypoint* proxy) handles L7 functionality (HTTP routing, L7 authorization, telemetry).

Compared to the sidecar mode, ambient mode reduces per-pod overhead (no sidecar container injected into every pod) and lets you adopt mesh features incrementally — first L4 via `ztunnel`, then L7 only for the namespaces that need it.

Components

The following components are used in the ambient mode:

- `ztunnel`: A DaemonSet that runs one pod per node and transparently tunnels traffic of the ambient workloads on that node over HTTP-Based Overlay Network Encapsulation (HBONE) with mutual TLS. It provides L4 features without a sidecar.
- **Waypoint proxy**: An optional L7 proxy deployed per namespace (or for a subset of its workloads and services). In DP, it is provisioned through the WaypointInstance custom resource, which is a DP abstraction on top of the native Istio waypoint provisioning. It adds declarative management of replicas (`Static`/`HPA`), resources (`Static`/`VPA`), node placement, and disruption budgets (PDB) per waypoint instance.

Prerequisites

To use the ambient mode, make sure to follow these requirements:

- Use Istio 1.25 or newer (ambient mode is available starting with Istio 1.25).
- Set `dataPlane.trafficRedirectionSetupMode` to `CNIPlugin`. Ambient mode requires the CNI plugin to set up traffic redirection.
- Enable ambient mode via the `ambient.enabled` module parameter.

Ambient mesh limitations

Ambient mode is compatible with federation and multicluster at the cluster level: you can enable ambient mode in a cluster that is a federation or multicluster member, and the existing inter-cluster interaction keeps working.

The limitation applies to individual workloads: federation and multicluster cover sidecar-mode workloads only. Cross-cluster traffic to and from a workload enrolled in ambient mode does not work in either direction.

⚠ To let a workload take part in a federation or a multicluster, keep it in sidecar mode — do not add the `istio.io/dataplane-mode=ambient` label to the workload or its namespace.

Both data plane modes can coexist in the same cluster, so you can keep cross-cluster workloads in sidecar mode and enroll the rest of them in ambient mode.

Enrolling workloads

The module installs and runs the ambient infrastructure (`ztunnel` and the waypoint controller for WaypointInstance resources), but it **does not** enroll your workloads automatically.

To enroll workloads, use the standard Istio labels:

1. Add the `istio.io/dataplane-mode=ambient` label to a namespace (or pod) to capture its traffic with `ztunnel` (L4).
2. Create a WaypointInstance resource in the namespace, then point workloads or services at it with the `istio.io/use-waypoint` label to enable L7 features.

See the “Examples” section for step-by-step configuration.

⚠ Delete all WaypointInstance resources before disabling ambient mode. With ambient mode disabled, the waypoint controller is not running and cannot reconcile or clean up waypoint resources.

Authentication

By default, the `user-authn` module provides authentication for Kiali. External authentication can also be configured via `externalAuthentication`. If both are disabled, the module uses basic auth with an auto-generated password.

To view the generated password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values istio -o json | jq '.istio.internal.auth.password'
```

To re-generate the password, delete the Secret:

```
d8 k -n d8-istio delete secret/kiali-basic-auth
```

⚠ The `auth.password` parameter is deprecated.

Estimating overhead

Using Istio will incur additional resource costs for both control-plane (istiod controller) and data-plane (istio-sidecars).

control-plane

The istiod controller continuously monitors the cluster configuration, compiles the settings for the istio-sidecars and distributes them over the network. Accordingly, the more applications and their instances, the more services, and the more frequently this configuration changes, the more computational resources are required and the greater the load on the network. Two approaches are supported to reduce the load on controller instances:

- horizontal scaling (module configuration `controlPlane.replicasManagement`) — the more controller instances, the fewer instances of istio-sidecars to serve for each controller and the less CPU and network load.
- data-plane segmentation using the Sidecar resource (recommended approach) — the smaller the scope of an individual istio-sidecar, the less data in the data-plane needs to be updated and the less CPU and network overhead.

A rough estimate of overhead for a control-plane instance that serves 1000 services and 2000 istio-sidecars is 1 vCPU and 1.5 GB RAM.

data-plane

The consumption of data-plane resources (istio-sidecar) is affected by many factors:

- number of connections,
- the intensity of requests,
- size of requests and responses,
- protocol (HTTP/TCP),
- number of CPU cores,
- complexity of Service Mesh configuration.

A rough estimate of the overhead for an istio-sidecar instance is 0.5 vCPU for 1000 requests/sec and 50 MB RAM. istio-sidecars also increase latency in network requests — about 2.5ms per request.

Third-party components

List of third-party software used in the `istio` module:

- Istio 1.25.2

  License: Apache License 2.0

  An open platform to connect, manage, and secure microservices.

- Istio 1.27.9

  License: Apache License 2.0

  An open platform to connect, manage, and secure microservices.

- Istio 1.29.6

  License: Apache License 2.0

  An open platform to connect, manage, and secure microservices.

- Kiali 2.7.1

  License: Apache License 2.0

  Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

- Kiali 2.12.0

License: Apache License 2.0

Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

- Kiali 2.21.0

  License: Apache License 2.0

  Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Available with limitations in: **Open/CE, Core, BE, SE, SE+, Certified Core/CSE Lite (1.73)**
>
> Available without limitations in: **Ultimate/EE**

> Included with limitations in extensions: **Advanced Networking, Advanced Infrastructure Security, Cluster Union, Multitenancy, Network Security**

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

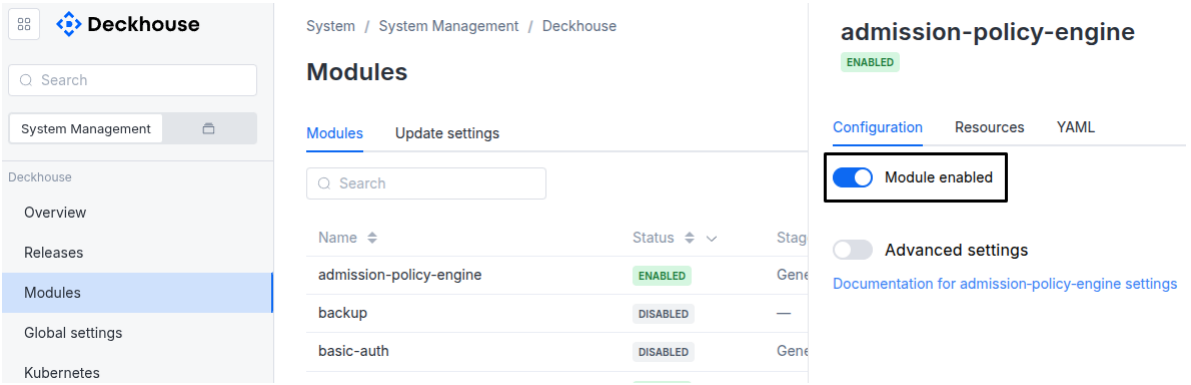
The module has 34 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `istio` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable istio
```

- **Using ModuleConfig `istio`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `istio` (create it if necessary);

  Example of a manifest to enable module `istio`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  enabled: true
```

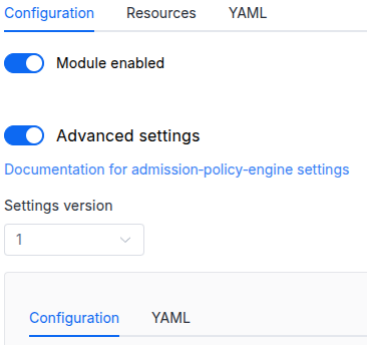
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `istio` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `istio` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `istio`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `istio` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc istio
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `istio` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `istio`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: istio
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the Deckhouse version: 1.72.0 and above.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `.settings.auth.password`.

- Updates from version **2** to **3**:
  - Move `.settings.enableHTTP10` to `.settings.dataPlane.enableHTTP10`.
  - Move `.settings.proxyConfig` to `.settings.dataPlane.proxyConfig`.

Parameters

Schema version: 3

- **settings**
  object
  - **settings.additionalVersions**
    array of strings

    Additional versions of Istio control plane to install. You can use specific namespace labels (`istio.io/rev=`) to switch between installed revisions.

    Default: `[]`

    Example:

    ```
    additionalVersions:
    - '1.25'
    ```

    - **Element of the array**
      string

      Allowed values: `1.25`, `1.27`, `1.29`

  - **settings.alliance**
    object

    Available in editions: Ultimate/EE

    Common options both for federation and multicluster.

    - **settings.alliance.ingressGateway**
      object

      Available in editions: Ultimate/EE

      Ingress gateway settings.

      - **settings.alliance.ingressGateway.advertise**
        array of objects

        Available in editions: Ultimate/EE

        The actual addresses that will be announced to remote clusters for organizing intercluster application requests. If not specified, the addresses will be discovered automatically.

        Default: `[]`

Examples:

```
advertise:
- address: 172.16.0.5
  port: 15443
```

```
advertise:
- address: somehost.example.com
  port: 15443
```

- **settings.alliance.ingressGateway.advertise.address**
  string

  Required value

  Pattern: `^([a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z]{2,}$`

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  Examples:

  ```
  address: somehost.example.com
  ```

  ```
  address: 172.16.0.5
  ```

- **settings.alliance.ingressGateway.advertise.port**
  integer

  Required value

  Allowed values: `1024 <= X <= 65535`

- **settings.alliance.ingressGateway.gatewayPodAnnotations**
  object

  Available in editions: Ultimate/EE

  Additional annotations for the ingress gateway DaemonSet pods.

  Example:

  ```
  gatewayPodAnnotations:
    checksum/config: xyz-123
  ```

- **settings.alliance.ingressGateway.inlet**
  string

  Available in editions: Ultimate/EE

  Method for exposing the ingress gateway:

  - `LoadBalancer`: For clusters whose cloud provider or infrastructure supports load balancers.
  - `NodePort`: For clusters without a load balancer.

  Default: `LoadBalancer`

  Allowed values: `LoadBalancer`, `NodePort`

  Example:

  ```
  inlet: LoadBalancer
  ```

- **settings.alliance.ingressGateway.loadBalancerClass**
  string

  Available in editions: Ultimate/EE

  Class of the load balancer for incoming network requests (passed to the `spec.loadBalancerClass` parameter of the provisioned service with the `LoadBalancer` type).

  Applies only when `inlet` is `LoadBalancer`.

- **settings.alliance.ingressGateway.nodePort**
  object

  Available in editions: Ultimate/EE

  Special settings for NodePort inlet.

  Examples:

  ```
  nodePort: {}
  ```

  ```
  nodePort:
    port: 30001
  ```

  - **settings.alliance.ingressGateway.nodePort.port**
    integer

    Static port number for NodePort-type Service. Must be in range, set by kube-apiserver –service-node-port-range argument (default is 30000-32767).

    Allowed values: `1024 <= X <= 65535`

- **settings.alliance.ingressGateway.nodeSelector**
  object

  Available in editions: Ultimate/EE

  Ingress gateway DaemonSet nodeSelector.

  The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  Example:

  ```
  nodeSelector:
    type: ingress
  ```

- **settings.alliance.ingressGateway.serviceAnnotations**
  object

  Available in editions: Ultimate/EE

  Additional service annotations. They can be used, e.g., for configuring a local LB in the Yandex Cloud (using the `yandex.cpi.flant.com/listener-subnet-id` annotation).

Example:

```
serviceAnnotations:
  yandex.cpi.flant.com/listener-subnet-id: xyz-123
```

- **settings.alliance.ingressGateway.tolerations**
  array of objects

  Available in editions: Ultimate/EE

  Ingress gateway DaemonSet tolerations.

  The same as `spec.tolerations` for the Kubernetes pod.

  Example:

  ```
  tolerations:
  - operator: Exists
  ```

  - **settings.alliance.ingressGateway.tolerations.effect**
    string
  - **settings.alliance.ingressGateway.tolerations.key**
    string
  - **settings.alliance.ingressGateway.tolerations.operator**
    string
  - **settings.alliance.ingressGateway.tolerations.tolerationSeconds**
    integer
  - **settings.alliance.ingressGateway.tolerations.value**
    string

- **settings.ambient**
  object

  Available in editions: Ultimate/EE

  Ambient mesh related settings.

  Default: `{"enabled":false}`

  - **settings.ambient.enabled**
    boolean

    Enable ambient mesh.

    Ambient-mode workloads cannot take part in federation or multicluster. Both remain functional for sidecar-mode workloads. See ambient mesh limitations.

    Default: `false`

    Example:

    ```
    enabled: true
    ```

  - **settings.ambient.waypointController**
    object

    Available in editions: Ultimate/EE

    Settings for CPU and memory requests and limits by waypoint-controller pods.

    - **settings.ambient.waypointController.resourcesManagement**
      object

      Settings for CPU and memory requests and limits by waypoint-controller pods.

      Examples:

      ```
      resourcesManagement:
        mode: VPA
        vpa:
          mode: InPlaceOrRecreate
          cpu:
            min: 25m
            max: 1
            limitRatio: 1.5
          memory:
            min: 64Mi
            max: 1Gi
            limitRatio: 1.5
      ```

      ```
      resourcesManagement:
        mode: Static
        static:
          requests:
            cpu: 25m
            memory: 64Mi
          limits:
            cpu: '1'
            memory: 1Gi
      ```

      - **settings.ambient.waypointController.resourcesManagement.mode**
        string

        Resource management mode for waypoint-controller pods.

        Possible values:

        - `Static`: Lets you set requests and limits manually in the `static` parameter section.
        - `VPA`: Lets VPA manage resources automatically. The mode is configured in the `vpa` parameter section.

        Default: `VPA`

        Allowed values: `VPA`, `Static`

      - **settings.ambient.waypointController.resourcesManagement.static**
        object

        Resource management options for the `Static` mode.

        - **settings.ambient.waypointController.resourcesManagement.static.limits**
          object

          Configuring CPU and memory limits.

          - **settings.ambient.waypointController.resourcesManagement.static.limits.cpu**
            string

Configuring CPU limits.

Pattern: `^[0-9]+m?$`

- **settings.ambient.waypointController.resourcesManagement.static.limits.memory**
  string or number

  Configuring memory limits.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.ambient.waypointController.resourcesManagement.static.requests**
  object

  Resource requests settings for pods.

  - **settings.ambient.waypointController.resourcesManagement.static.requests.cpu**
    string

    Configuring CPU requests.

    Pattern: `^[0-9]+m?$`

  - **settings.ambient.waypointController.resourcesManagement.static.requests.memory**
    string or number

    Configuring memory requests.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.ambient.waypointController.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.ambient.waypointController.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.ambient.waypointController.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      This ratio is used for calculating the initial CPU limits for a pod.

      If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

    - **settings.ambient.waypointController.resourcesManagement.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `1`

      Pattern: `^[0-9]+m?$`

    - **settings.ambient.waypointController.resourcesManagement.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `25m`

      Pattern: `^[0-9]+m?$`

  - **settings.ambient.waypointController.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **settings.ambient.waypointController.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      This ratio is used for calculating the initial memory limits for a pod.

      If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

    - **settings.ambient.waypointController.resourcesManagement.vpa.memory.max**
      string or number

      The maximum memory requests the VPA can set.

      Default: `1Gi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.ambient.waypointController.resourcesManagement.vpa.memory.min**
      string or number

      The minimum memory requests the VPA can set.

      Default: `64Mi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.ambient.waypointController.resourcesManagement.vpa.mode**
    string

    VPA operating mode.

    Possible values:

    - `Initial`: VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
    - `InPlaceOrRecreate`: VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.

    Default: `InPlaceOrRecreate`

    Allowed values: `Initial`, `InPlaceOrRecreate`

- **settings.auth**
  object

  Options related to authentication or authorization in the application.

  Example:

```
auth:
  externalAuthentication:
    authURL: https://dex.d8.svc.cluster.local/dex/auth
    authSignInURL: https://example.com/dex/sign_in
  allowedUserGroups:
  - admins
```

- **settings.auth.allowedUserEmails**
  array of strings

  An array of emails of users that can access module's public web interfaces.

  This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

- **settings.auth.allowedUserGroups**
  array of strings

  An array of user groups that can access module's public web interfaces.

  This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

  Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

- **settings.auth.externalAuthentication**
  object

  Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

  > **i** External authentication is enabled automatically if the user-authn module is enabled.

  - **settings.auth.externalAuthentication.authSignInURL**
    string

    The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

    Example:

    ```
    authSignInURL: https://example.com/dex/sign_in
    ```

  - **settings.auth.externalAuthentication.authURL**
    string

    The URL of the authentication service.

    If the user is authenticated, the service should return an HTTP 200 response code.

    Example:

    ```
    authURL: https://example.com/dex/auth
    ```

- **settings.auth.satisfyAny**
  boolean

  Enables single authentication.

  If used together with the whitelistSourceRanges parameter, it authorizes all the users from above networks (no need to enter a username and password).

  Default: `false`

  Example:

  ```
  satisfyAny: true
  ```

- **settings.auth.whitelistSourceRanges**
  array of strings

  An array if CIDRs that are allowed to authenticate in module's public web interfaces.

  Example:

  ```
  whitelistSourceRanges:
  - 1.1.1.1/32
  ```

- **settings.ca**
  object

  Explicitly specified root certificate. It signs individual service certificates to use in mutual TLS connections.

  The certificate can be provided in one of two ways:

  - inline via the `cert` and `key` fields (with optional `chain` and `root`);
  - by referencing an existing Secret via the `secretRef` field.

  These two ways are mutually exclusive.

  Regardless of how it is provided, the CA is validated the same way: the signing certificate must be a single CA certificate (`basicConstraints = CA:TRUE`) whose key matches it, and, when an intermediate CA is used, the `root` must actually anchor the `cert` through the `chain`. Invalid material hard-blocks the module rather than being pushed to istiod.

  Removing `cert/secretRef` from the configuration does **not** revert the mesh to an auto-generated self-signed CA: the module keeps the last-published CA so an already-working mesh is never rotated implicitly — rotating the mesh root breaks mTLS cluster-wide until every workload re-trusts it. To deliberately return to a self-signed CA, remove the `cert/secretRef` configuration, delete the `d8-istio/cacerts` Secret **and** restart the Deckhouse pod. The module then generates a fresh self-signed CA. This is a mesh-wide trust change and triggers a rollout.

  - **settings.ca.cert**
    string

    The root or intermediate certificate in PEM format.

  - **settings.ca.chain**
    string

    A certificate chain in PEM format if `cert` is an intermediate certificate.

  - **settings.ca.key**
    string

    The key to the root certificate in PEM format.

  - **settings.ca.root**
    string

    The root certificate in PEM format if `cert` is an intermediate certificate. Setting `root` to the same certificate as `cert` explicitly anchors mesh trust at that certificate; this is allowed even for a non-self-signed intermediate, but rotating it later changes the mesh root (a mesh-wide trust event).

- **settings.ca.secretRef**
  object

  Reference to an existing Secret that contains the CA certificate and key.

  The Secret may be in one of the following formats (auto-detected):

  - a cert-manager `kubernetes.io/tls` Secret with the `tls.crt`, `tls.key` and `ca.crt` keys — this allows issuing the CA with any preferred tooling (cert-manager self-signed issuer, Vault issuer, etc.);
  - a native Istio `cacerts` Secret with the `ca-cert.pem`, `ca-key.pem`, `cert-chain.pem` and `root-cert.pem` keys.

  The signing certificate must be a single CA certificate (`basicConstraints = CA:TRUE`) whose key matches it. For cert-manager, issue it with `isCA: true`.

  If the referenced Secret is missing or malformed (not valid PEM, not a CA certificate, or a mismatched key pair) on the first resolution of this `secretRef`, the module will not render until the issue is fixed (it will not silently fall back to a self-signed CA or to any previously-used CA). If the same `secretRef` was already resolved successfully and later becomes unavailable or invalid, the module keeps using that last successfully-resolved CA so an already-working mesh is not regressed. Changing `secretRef` to point at a different Secret is treated as a first resolution: if the new target cannot be resolved, the module hard-blocks instead of keeping the old CA.

  Example:

  ```
  secretRef:
    name: my-istio-ca
    namespace: my-pki
  ```

  - **settings.ca.secretRef.name**
    string

    Required value

    Name of the Secret containing the CA certificate and key.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **settings.ca.secretRef.namespace**
    string

    Namespace of the Secret. If not specified, the Secret is looked up in the module namespace.

    Default: `d8-istio`

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.configAnalysis**
  object

  Available in editions: Ultimate/EE

  Periodic Istio configuration analysis (`istioctl analyze`) that exports findings as Prometheus metrics and alerts.

  Default: `{"enabled":true}`

  - **settings.configAnalysis.enabled**
    boolean

    Enable the Istio configuration analyzer Deployment for each installed control-plane revision.

    Set to `false` to disable analysis and remove related workloads.

    Default: `true`

    Example:

    ```
    enabled: false
    ```

- **settings.controlPlane**
  object

  istiod specific settings.

  - **settings.controlPlane.extraEnvs**
    object

    Additional environment variables for the istiod (pilot) container.

    Keys reserved by the module cannot be overridden:

    - `ISTIO_MULTIROOT_MESH`
    - `ENABLE_ENHANCED_RESOURCE_SCOPING`
    - `PILOT_HTTP10`
    - `PILOT_ENABLE_AMBIENT`

    Example:

    ```
    extraEnvs:
      GODEBUG: gctrace=1
    ```

  - **settings.controlPlane.nodeSelector**
    object

    Optional `nodeSelector` for istiod. The same as the `spec.nodeSelector` pod parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.controlPlane.replicasManagement**
    object

    Replication management settings and scaling of istiod.

    Examples:

    ```
    replicasManagement:
      mode: Standard
    ```

    ```
    replicasManagement:
      mode: Static
      static:
        replicas: 3
    ```

```
replicasManagement:
  mode: HPA
  hpa:
    minReplicas: 2
    maxReplicas: 5
    metrics:
    - type: CPU
      targetAverageUtilization: 80
```

- **settings.controlPlane.replicasManagement.hpa**
  object

  Options for replicas management for the `HPA` mode.

  - **settings.controlPlane.replicasManagement.hpa.maxReplicas**
    number

    Required value

    The upper limit for the number of replicas to which the HPA can scale up. It cannot be less that `minReplicas`.

    Allowed values: `1 <= X`

  - **settings.controlPlane.replicasManagement.hpa.metrics**
    array of objects

    Required value

    The HPA will use these metrics to decide whether to increase or decrease the number of replicates.

    - **settings.controlPlane.replicasManagement.hpa.metrics.targetAverageUtilization**
      number

      Required value

      The target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

      Allowed values: `1 <= X <= 100`

    - **settings.controlPlane.replicasManagement.hpa.metrics.type**
      string

      Required value

      Metric type.

      Allowed values: `CPU`

  - **settings.controlPlane.replicasManagement.hpa.minReplicas**
    number

    Required value

    The lower limit for the number of replicas to which the HPA can scale down.

    Allowed values: `1 <= X`

- **settings.controlPlane.replicasManagement.mode**
  string

  Replicas management mode:

  - `Standard` — replicas management and scaling mode according to the global fault tolerance mode (the highAvailability parameter);
  - `Static` — the mode, where the number of replicas is specified explicitly (the static.replicas parameter);
  - `HPA` — the mode, where the number of replicas is calculated automatically using HPA based on CPU usage. You can configure this mode by modifying parameters in the hpa parameter section.

  Default: `Standard`

  Allowed values: `Standard`, `Static`, `HPA`

- **settings.controlPlane.replicasManagement.static**
  object

  Options for replicas management for the `Static` mode.

  - **settings.controlPlane.replicasManagement.static.replicas**
    number

    Required value

    Desired number of replicas.

    Allowed values: `1 <= X`

- **settings.controlPlane.resourcesManagement**
  object

  Settings for CPU and memory requests and limits by istiod pods.

  Examples:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: '2'
      memory: 2Gi
```

```
resourcesManagement:
 mode: VPA
 vpa:
  mode: Auto
  cpu:
   min: 50m
   max: 2
   limitRatio: 1.5
  memory:
   min: 256Mi
   max: 2Gi
   limitRatio: 1.5
```

- **settings.controlPlane.resourcesManagement.mode**
  string

  Resource management mode:

  - `Static` — allows you to specify requests/limits. The parameters of this mode are defined in the static parameter section;
  - `VPA` — uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

  Default: `VPA`

  Allowed values: `VPA`, `Static`

- **settings.controlPlane.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.controlPlane.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.controlPlane.resourcesManagement.static.limits.cpu**
      string

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.controlPlane.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.controlPlane.resourcesManagement.static.requests.cpu**
      string

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.controlPlane.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.controlPlane.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      This ratio is used for calculating the initial CPU limits for a pod.

      If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

    - **settings.controlPlane.resourcesManagement.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `2`

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `50m`

      Pattern: `^[0-9]+m?$`

  - **settings.controlPlane.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **settings.controlPlane.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      This ratio is used for calculating the initial memory limits for a pod.

      If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

    - **settings.controlPlane.resourcesManagement.vpa.memory.max**
      string or number

      The maximum memory requests the VPA can set.

Default: `2Gi`

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.vpa.memory.min**
  string or number

  The minimum memory requests the VPA can set.

  Default: `256Mi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.vpa.mode**
  string

  VPA operating mode.

  - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
  - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
  - `Auto` — VPA automatically recreates pods to apply updated resource values.

  > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

  Default: `InPlaceOrRecreate`

  Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.controlPlane.tolerations**
  array of objects

  Optional `tolerations` for istiod. The same as `spec.tolerations` for the Kubernetes pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.controlPlane.tolerations.effect**
    string
  - **settings.controlPlane.tolerations.key**
    string
  - **settings.controlPlane.tolerations.operator**
    string
  - **settings.controlPlane.tolerations.tolerationSeconds**
    integer
  - **settings.controlPlane.tolerations.value**
    string

- **settings.dataPlane**
  object
  - **settings.dataPlane.accessLog**
    object
    - **settings.dataPlane.accessLog.jsonLabels**
      object

      Structured access log format using key-value pairs. Template operators are described in envoy documentation.

      Default: `{"start time":"%START TIME%","method":"%REQ(:METHOD)%","path":"%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%","protocol":"%PROTOCOL%","response code":"%RESPONSE CODE%","response flags":"%RESPONSE_FLAGS%","bytes received":"%BYTES RECEIVED%","bytes sent":"%BYTES SENT%","duration":"%DURATION%","user_agent":"%REQ(USER-AGENT)%","authority":"%REQ(:AUTHORITY)%","upstream_host":"%UPSTREAM_HOST%"}`

    - **settings.dataPlane.accessLog.textFormat**
      string

      Sidecar's access log format template. Template operators are described in envoy documentation.

      Default: `[%START TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%" %RESPONSE CODE% %RESPONSE_FLAGS% %RESPONSE_CODE_DETAILS% %CONNECTION_TERMINATION_DETAILS% "%UPSTREAM_TRANSPORT_FAILURE_REASON%" %BYTES RECEIVED% %BYTES SENT% %DURATION% %RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%" "%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM_HOST%" %UPSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_REMOTE_ADDRESS% %REQUESTED_SERVER_NAME% %ROUTE_NAME%`

    - **settings.dataPlane.accessLog.type**
      string

      Default: `Text`

      Allowed values: `Text`, `JSON`

  - **settings.dataPlane.enableHTTP10**
    boolean

    Whether to handle HTTP/1.0 requests in istio-sidecars or deny them with `426 Upgrade Required` response.

    Default: `false`

    Example:

    ```
    enableHTTP10: true
    ```

  - **settings.dataPlane.extensionProviders**
    array of objects

    Additional Istio extension providers to register in the mesh configuration.

    In each provider, set exactly one of `envoyExtAuthzGrpc`, `envoyExtAuthzHttp`, `opentelemetry` or `zipkin`.

    The module does not enable the registered providers. Reference them by name in your AuthorizationPolicy (`action: CUSTOM`) or Telemetry. Mesh-wide tracing configured in `tracing` is not affected.

    The `d8-main` and `deckhouse-tracing` provider names are reserved for the module. The `envoy`, `prometheus` and `stackdriver` names are reserved for the providers built into Istio, because a provider with such a name would replace the built-in one.

    Extension provider settings.

    Default: `[]`

    Examples:

    ```
    extensionProviders:
    - name: authservice-grpc
      envoyExtAuthzGrpc:
        service: authservice.example.svc.cluster.local
        port: 10003
    ```

```
extensionProviders:
- name: oauth2-proxy
  envoyExtAuthzHttp:
    service: oauth2-proxy.example.svc.cluster.local
    port: 4180
    includeRequestHeadersInCheck:
    - authorization
    - cookie
    headersToUpstreamOnAllow:
    - authorization
    - x-auth-request-user
    headersToDownstreamOnDeny:
    - content-type
    - set-cookie
- name: otel-tracing
  opentelemetry:
    service: opentelemetry-collector.observability.svc.cluster.local
    port: 4317
- name: zipkin-tracing
  zipkin:
    service: zipkin.example.svc.cluster.local
    port: 9411
```

- **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc**
  object

  External authorization provider that implements Envoy ext_authz gRPC API.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.clearRouteCache**
    boolean

    Clear route cache to let the external authorization service affect routing decisions.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.failOpen**
    boolean

    Allow requests when the authorization service is unavailable or returns a 5xx error.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.port**
    integer

    Required value

    Service port.

    Allowed values: `1 <= X <= 65535`

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.service**
    string

    Required value

    Service that implements Envoy ext_authz gRPC API. The format is `[<namespace>/]<hostname>`.

    `<hostname>` must be the full hostname of a Service, including the cluster domain (for example, `authservice.example.svc.cluster.local`), or a ServiceEntry host. Short names and IP addresses do not work. Specify `<namespace>` only if this hostname exists in several namespaces.

    If Istio cannot find the service, the requests that match AuthorizationPolicy with `action: CUSTOM` referencing this provider are denied.

    Pattern: `^([a-z0-9]([-a-z0-9]{0,61}[a-z0-9])?/)?([a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?\.)*[a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?$`

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.statusOnError**
    string

    HTTP status returned to the client when communication with the authorization service fails.

    Only the status codes that Envoy supports are allowed, for example `403` or `503`.

    Pattern: `^(100|20[0-8]|226|30[0-578]|4(0[0-9]|1[0-7]|2[1-4689]|31)|5(0[0-8]|1[01]))$`

    Example:

    ```
    statusOnError: '503'
    ```

  - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.timeout**
    string

    Maximum duration the proxy waits for the authorization service response.

    Pattern: `^([0-9]+(\.[0-9]+)?(ns|us|µs|ms|s|m|h))+$`

    Example:

    ```
    timeout: 5s
    ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp**
  object

  External authorization provider that implements Envoy ext_authz HTTP API.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.clearRouteCache**
    boolean

    Clear route cache to let the external authorization service affect routing decisions.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.failOpen**
    boolean

    Allow requests when the authorization service is unavailable or returns a 5xx error.

  - **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.headersToDownstreamOnAllow**
    array of strings

    Headers from the authorization service response to add or override in the response sent back to the client when the request is allowed.

    Exact, prefix (`abc*`) and suffix (`*abc`) matches are supported.

    Example:

    ```
    headersToDownstreamOnAllow:
    - set-cookie
    ```

  - **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.headersToDownstreamOnDeny**
    array of strings

Headers from the authorization service response to send back to the client when the request is denied.

Exact, prefix (`abc*`) and suffix (`*abc`) matches are supported.

Example:

```
headersToDownstreamOnDeny:
- content-type
- set-cookie
```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.headersToUpstreamOnAllow**
  array of strings

  Headers from the authorization service response to add or override in the request sent upstream when the request is allowed.

  Exact, prefix (`abc*`) and suffix (`*abc`) matches are supported.

  Example:

  ```
  headersToUpstreamOnAllow:
  - authorization
  - x-auth-request-user
  ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.includeAdditionalHeadersInCheck**
  object

  Additional headers with fixed values to send to the authorization service.

  A header with the same name in the original request is overridden.

  Example:

  ```
  includeAdditionalHeadersInCheck:
    x-ext-authz-source: istio
  ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.includeRequestHeadersInCheck**
  array of strings

  Headers of the original request to send to the authorization service.

  The `Host`, `Method`, `Path` and `Content-Length` headers are always sent.

  Exact, prefix (`abc*`) and suffix (`*abc`) matches are supported.

  Example:

  ```
  includeRequestHeadersInCheck:
  - authorization
  - cookie
  ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.pathPrefix**
  string

  Prefix added to the path of the original request before sending it to the authorization service.

  For example, with `pathPrefix: /check` a request to `/foo` is checked as `/check/foo`.

  Pattern: `^/`

  Example:

  ```
  pathPrefix: "/check"
  ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.port**
  integer

  Required value

  Service port.

  Allowed values: `1 <= X <= 65535`

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.service**
  string

  Required value

  Service that implements Envoy ext_authz HTTP API. The format is `[<namespace>/]<hostname>`.

  `<hostname>` must be the full hostname of a Service, including the cluster domain (for example, `oauth2-proxy.example.svc.cluster.local`), or a ServiceEntry host. Short names and IP addresses do not work. Specify `<namespace>` only if this hostname exists in several namespaces.

  If Istio cannot find the service, the requests that match AuthorizationPolicy with `action: CUSTOM` referencing this provider are denied.

  Pattern: `^([a-z0-9]([-a-z0-9]{0,61}[a-z0-9])?/)?([a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?\.)*[a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?$`

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.statusOnError**
  string

  HTTP status returned to the client when communication with the authorization service fails.

  Only the status codes that Envoy supports are allowed, for example `403` or `503`.

  Pattern: `^(100|20[0-8]|226|30[0-578]|4(0[0-9]|1[0-7]|2[1-4689]|31)|5(0[0-8]|1[01]))$`

  Example:

  ```
  statusOnError: '503'
  ```

- **settings.dataPlane.extensionProviders.envoyExtAuthzHttp.timeout**
  string

  Maximum duration the proxy waits for the authorization service response.

  Pattern: `^([0-9]+(\.[0-9]+)?(ns|us|µs|ms|s|m|h))+$`

  Example:

  ```
  timeout: 5s
  ```

- **settings.dataPlane.extensionProviders.name**
  string

  Required value

  Unique extension provider name.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.dataPlane.extensionProviders.opentelemetry**
  object

  Tracing provider that exports spans with OpenTelemetry Protocol (OTLP).

  Spans are exported over gRPC unless `http` is set.

  - **settings.dataPlane.extensionProviders.opentelemetry.grpc**
    object

    Settings for exporting spans with OTLP over gRPC.

    Cannot be used together with `http`.

    - **settings.dataPlane.extensionProviders.opentelemetry.grpc.initialMetadata**
      array of objects

      Additional metadata to send with each gRPC stream.

      Example:

      ```
      initialMetadata:
      - name: x-scope-orgid
        value: tenant-a
      ```

      - **settings.dataPlane.extensionProviders.opentelemetry.grpc.initialMetadata.name**
        string

        Required value

      - **settings.dataPlane.extensionProviders.opentelemetry.grpc.initialMetadata.value**
        string

        Required value

    - **settings.dataPlane.extensionProviders.opentelemetry.grpc.timeout**
      string

      Maximum duration the proxy waits for the collector response.

      Pattern: `^([0-9]+(\.[0-9]+)?(ns|us|µs|ms|s|m|h))+$`

      Example:

      ```
      timeout: 10s
      ```

  - **settings.dataPlane.extensionProviders.opentelemetry.http**
    object

    Settings for exporting spans with OTLP over HTTP.

    Cannot be used together with `grpc`.

    - **settings.dataPlane.extensionProviders.opentelemetry.http.headers**
      array of objects

      Additional headers to send with each export request.

      Example:

      ```
      headers:
      - name: x-scope-orgid
        value: tenant-a
      ```

      - **settings.dataPlane.extensionProviders.opentelemetry.http.headers.name**
        string

        Required value

      - **settings.dataPlane.extensionProviders.opentelemetry.http.headers.value**
        string

        Required value

    - **settings.dataPlane.extensionProviders.opentelemetry.http.path**
      string

      Required value

      Path of the OTLP traces endpoint.

      Example:

      ```
      path: "/v1/traces"
      ```

    - **settings.dataPlane.extensionProviders.opentelemetry.http.timeout**
      string

      Maximum duration the proxy waits for the collector response.

      Pattern: `^([0-9]+(\.[0-9]+)?(ns|us|µs|ms|s|m|h))+$`

      Example:

      ```
      timeout: 10s
      ```

  - **settings.dataPlane.extensionProviders.opentelemetry.maxTagLength**
    integer

    Maximum length of the request path to extract and include in the `http.url` span tag. If not set, the Envoy default (`256`) is used.

    Allowed values: `1 <= X`

  - **settings.dataPlane.extensionProviders.opentelemetry.port**
    integer

    Required value

    OpenTelemetry collector port.

    Allowed values: `1 <= X <= 65535`

    Example:

    ```
    port: 4317
    ```

  - **settings.dataPlane.extensionProviders.opentelemetry.resourceDetectors**
    object

OpenTelemetry resource detectors that add attributes to the exported spans.

- **settings.dataPlane.extensionProviders.opentelemetry.resourceDetectors.dynatrace**
  object

  Read resource attributes from the files written by Dynatrace OneAgent.

  Set to `{}` to enable.

- **settings.dataPlane.extensionProviders.opentelemetry.resourceDetectors.environment**
  object

  Read resource attributes from the `OTEL_RESOURCE_ATTRIBUTES` environment variable of the proxy.

  Set to `{}` to enable.

- **settings.dataPlane.extensionProviders.opentelemetry.service**
  string

  Required value

  OpenTelemetry collector service. The format is `[<namespace>/]<hostname>`.

  `<hostname>` must be the full hostname of a Service, including the cluster domain (for example, `opentelemetry-collector.observability.svc.cluster.local`), or a ServiceEntry host. Short names and IP addresses do not work. Specify `<namespace>` only if this hostname exists in several namespaces.

  If Istio cannot find the service, spans are not exported.

  Pattern: `^([a-z0-9]([-a-z0-9]{0,61}[a-z0-9])?/)?([a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?\.)*[a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?$`

  Example:

  ```
  service: opentelemetry-collector.observability.svc.cluster.local
  ```

- **settings.dataPlane.extensionProviders.zipkin**
  object

  Tracing provider that exports spans with Zipkin API.

  - **settings.dataPlane.extensionProviders.zipkin.enable64bitTraceId**
    boolean

    Use 64-bit trace IDs instead of 128-bit ones.

  - **settings.dataPlane.extensionProviders.zipkin.maxTagLength**
    integer

    Maximum length of the request path to extract and include in the `http.url` span tag. If not set, the Envoy default (`256`) is used.

    Allowed values: `1 <= X`

  - **settings.dataPlane.extensionProviders.zipkin.path**
    string

    Path of the Zipkin spans endpoint. If not set, `/api/v2/spans` is used.

    Pattern: `^/`

    Example:

    ```
    path: "/api/v2/spans"
    ```

  - **settings.dataPlane.extensionProviders.zipkin.port**
    integer

    Required value

    Zipkin collector port.

    Allowed values: `1 <= X <= 65535`

    Example:

    ```
    port: 9411
    ```

  - **settings.dataPlane.extensionProviders.zipkin.service**
    string

    Required value

    Zipkin collector service. The format is `[<namespace>/]<hostname>`.

    `<hostname>` must be the full hostname of a Service, including the cluster domain (for example, `zipkin.example.svc.cluster.local`), or a ServiceEntry host. Short names and IP addresses do not work. Specify `<namespace>` only if this hostname exists in several namespaces.

    If Istio cannot find the service, spans are not exported.

    Pattern: `^([a-z0-9]([-a-z0-9]{0,61}[a-z0-9])?/)?([a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?\.)*[a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?$`

    Example:

    ```
    service: zipkin.example.svc.cluster.local
    ```

- **settings.dataPlane.proxyConfig**
  object

  Mesh-wide defaults for ProxyConfig configurations.

  - **settings.dataPlane.proxyConfig.holdApplicationUntilProxyStarts**
    boolean

    With this feature, the sidecar-injector injects the sidecar at the first place of Pod’s container list and adds a postStart hook to be sure if the Envoy proxy is initialized before the application. So the Envoy is able to handle requests without application network errors.

    This global flag can be overriden per Pod by an annotation — `proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts": true }'`.

    Has no effect on native sidecars, which Istio 1.27 and newer inject by default: they always start before the application.

    Default: `false`

    Example:

    ```
    holdApplicationUntilProxyStarts: true
    ```

  - **settings.dataPlane.proxyConfig.idleTimeout**
    string

    Timeout for connections without application activity established between the client’s istio-sidecar and the service. When the timeout expires, the connection between the sidecar and the service is closed, but the connection between the application and the sidecar is not closed. If set to `0s`,

the timeout is disabled.

This global flag can be overriden per Pod by an annotation:

```
proxy.istio.io/config: |-
  proxyMetadata:
    ISTIO_META_IDLE_TIMEOUT: "12h"
```

> **Warning!** Disabling this timeout (setting the value to `0s`) is very likely to result in leaky connections due to TCP FIN packet loss, etc.
> **Warning!** After changing this setting, a restart of the client pods is required.

Default: `1h`

Pattern: `^[0-9]+(s|m|h)$`

Example:

```
idleTimeout: 24h
```

- **settings.dataPlane.trafficRedirectionSetupMode**
  string

  Managing the redirection mode of application traffic to be forwarded under Istio control in the Pod's network namespace.

  - `CNIPlugin` — in this mode, the configuration is performed by a CNI plugin when creating a Pod on a node. This mode does not require additional permissions for Pods and is recommended. This mode has limitations when using application init-containers that perform network communication with other services. The limitations apply only to regular sidecars, which Istio 1.25 injects by default.
  - `InitContainer` — classic mode, each application Pod is automatically injected with a special init-container that configures the network environment of the Pod. In order to perform this configuration, the init-container is given additional permissions, which may not meet the security requirements of individual installations.

  Default: `InitContainer`

  Allowed values: `CNIPlugin`, `InitContainer`

  Examples:

  ```
  trafficRedirectionSetupMode: CNIPlugin
  ```

  ```
  trafficRedirectionSetupMode: InitContainer
  ```

- **settings.federation**
  object

  Available in editions: Ultimate/EE

  Parameters for federating with other clusters.

  - **settings.federation.enabled**
    boolean

    Designate this cluster as a federation member. See how to enable federation.

    Federation covers sidecar-mode workloads only. Ambient-mode workloads cannot take part in it. See ambient mesh limitations.

    Default: `false`

- **settings.globalVersion**
  string

  Specific version of Istio control-plane which handles unspecific versions of data plane (namespaces with `istio-injection=enabled` label, not `istio.io/rev=`).

  Default: `1.27`

  Allowed values: `1.25`, `1.27`, `1.29`

  Example:

  ```
  globalVersion: '1.29'
  ```

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

  Example:

  ```
  highAvailability: true
  ```

- **settings.https**
  object

  What certificate type to use with module's public web interfaces.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**

object
- **settings.https.customCertificate.secretName**
  string

  The name of the secret in the `d8-system` namespace to use with Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

  This secret must have the kubernetes.io/tls format.

  Default: `false`

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will use HTTPS and get a certificate from the clusterissuer defined in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will use HTTPS using the certificate from the `d8-system` namespace.
  - `OnlyInURI` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

  Caution! Unlike other modules, Istio doesn't support non-secured HTTP (`mode: Disabled`).

  Default: `CertManager`

  Allowed values: `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for Kiali, metadata-exporter and proxy-api.

  Optional. By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.jwksResolverAdditionalRootCA**
  string

  An additional root certificate in PEM format. It is used by istiod when resolving JWKS URIs over HTTPS.

  Example: —–BEGIN CERTIFICATE—– MIIDXTCCAkWgAwIBAgIJAN… —–END CERTIFICATE—–

- **settings.multicluster**
  object

  Available in editions: Ultimate/EE

  Multicluster parameters.

  - **settings.multicluster.enabled**
    boolean

    Designate this cluster as a multicluster member. See how to enable multicluster.

    Multicluster covers sidecar-mode workloads only. Ambient-mode workloads cannot take part in it. See ambient mesh limitations.

    Default: `false`

    Example:

    ```
    enabled: true
    ```

- **settings.nodeSelector**
  object

  Optional `nodeSelector` for istio-operator, metadata-exporter and Kiali. The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.outboundTrafficPolicyMode**
  string

  Policy for requests directed to external services that aren't registered in the mesh.

  With `RegistryOnly`, external services must be registered via a `ServiceEntry` custom resource or routed through an `egressgateway`.

  Default: `AllowAny`

  Allowed values: `AllowAny`, `RegistryOnly`

  Example:

  ```
  outboundTrafficPolicyMode: AllowAny
  ```

- **settings.sidecar**
  object

  Network settings for traffic capture by istio sidecar.

  - **settings.sidecar.excludeInboundPorts**
    array of strings

    The range of inbound ports whose traffic is guaranteed not to flow through Istio.

    You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeInboundPorts` annotation.

    Default: `[]`

    Example:

    ```
    excludeInboundPorts:
    - '8080'
    - '8443'
    ```

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,5}$`

  - **settings.sidecar.excludeOutboundIPRanges**
    array of strings

    Traffic to these IP ranges is guaranteed not to flow through Istio.

    You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeOutboundIPRanges` annotation.

    Default: `[]`

    Example:

```
excludeOutboundIPRanges:
- 10.1.1.0/24
```

- **Element of the array**
  string

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **settings.sidecar.excludeOutboundPorts**
  array of strings

  The range of outbound ports whose traffic is guaranteed not to flow through Istio.

  You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeOutboundPorts` annotation.

  Default: `[]`

  Example:

```
excludeOutboundPorts:
- '8080'
- '8443'
```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,5}$`

- **settings.sidecar.includeOutboundIPRanges**
  array of strings

  Traffic to these IP ranges is forcibly routed through Istio.

  You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/includeOutboundIPRanges` annotation.

  Default: `["0.0.0.0/0"]`

  Example:

```
includeOutboundIPRanges:
- 10.1.1.0/24
```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **settings.sidecar.resourcesManagement**
  object

  Manages Istio sidecar container resources.

  Caution! The setting only applies to new Pods with `istio-proxy`.

  Example:

```
resourcesManagement:
 static:
   requests:
     cpu: 100m
     memory: 128Mi
   limits:
     cpu: 2000m
     memory: 1Gi
```

  - **settings.sidecar.resourcesManagement.mode**
    string

    Resource management mode:

    - `Static` — allows you to specify requests/limits. The parameters of this mode are defined in the static parameter section;

    Default: `Static`

    Allowed values: `Static`

  - **settings.sidecar.resourcesManagement.static**
    object

    Resource management options for the `Static` mode.

    - **settings.sidecar.resourcesManagement.static.limits**
      object

      Configuring CPU and memory limits.

      - **settings.sidecar.resourcesManagement.static.limits.cpu**
        string

        Configuring CPU limits.

        Default: `2000m`

        Pattern: `^[0-9]+m?$`

      - **settings.sidecar.resourcesManagement.static.limits.memory**
        string or number

        Configuring memory limits.

        Default: `1Gi`

        Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.sidecar.resourcesManagement.static.requests**
      object

      Resource requests settings for pods.

      - **settings.sidecar.resourcesManagement.static.requests.cpu**
        string

        Configuring CPU requests.

        Default: `100m`

        Pattern: `^[0-9]+m?$`

      - **settings.sidecar.resourcesManagement.static.requests.memory**
        string or number

Configuring memory requests.

Default: `128Mi`

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.telemetryAPI**
  object

  Telemetry API (`telemetry.istio.io`) integration.

  When `enabled` is `true`, the module drives mesh metrics via the Telemetry API and `meshConfig.defaultProviders.metrics: [prometheus]`, keeps `values.telemetry.enabled` but disables the previous `telemetry.v2` filters, and extends the always-present `Telemetry d8-main` in `d8-istio` with `spec.metrics`. When `tracing.enabled` is `true` with a configured `tracing.collector`, `spec.tracing` is added via `deckhouse-tracing` so Grafana and Kiali keep seeing `istio_*` series from sidecars.

  When `false` (default), the previous `telemetry.v2` stack stays on (including Sail's `telemetry.v2.prometheus`), without `defaultProviders.metrics`, while `Telemetry d8-main` still enables access logs (backwards compatible with releases since the access-log `Telemetry` was introduced).

  `dataPlane.accessLog` defines the Envoy log template in `meshConfig.extensionProviders`; the `Telemetry d8-main` object selects that provider for access logging.

  - **settings.telemetryAPI.enabled**
    boolean

    Telemetry API path: wires `defaultProviders.metrics`, turns off `telemetry.v2`, adds `spec.metrics` (and tracing when configured) to `Telemetry d8-main`.

    Default: `false`

    Example:

    ```
    enabled: true
    ```

- **settings.tolerations**
  array of objects

  Optional `tolerations` for istio-operator, metadata-exporter and Kiali. The same as `spec.tolerations` for the Kubernetes pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string
- **settings.tracing**
  object

  Tracing parameters.

  If the `ingress-nginx` module is enabled (`global.enabledModules` contains `ingress-nginx`), the module creates Istio `Telemetry` resource `ingress-nginx-disable-span-reporting` in namespace `d8-ingress-nginx`. Built-in Ingress controller pods with an Istio sidecar stop exporting spans to the tracing backend (reduces noise while keeping meshes functional).

  - **settings.tracing.collector**
    object

    Tracing collection settings. Single source for both legacy control-plane tracing and the Telemetry API path.

    - If `telemetryAPI.enabled` is `false` and `tracing.enabled` is `true`, the module fills `meshConfig.defaultConfig.tracing.zipkin` from `collector.zipkin.address` (with `tracing.sampling`). OpenTelemetry export requires the Telemetry API path.
    - If `telemetryAPI.enabled` is `true` and tracing is enabled, configure either `collector.opentelemetry` or `collector.zipkin`. When both are set, OpenTelemetry wins. In either case the extension provider is named `deckhouse-tracing` with `spec.tracing` on `Telemetry d8-main`.
    - **settings.tracing.collector.opentelemetry**
      object

      OpenTelemetry extension provider for the Telemetry API path (`meshConfig.extensionProviders` + mesh-wide `spec.tracing`).

      Requires `service` and `port` (gRPC OTLP by default; set `http.path` for HTTP export). Not used when `telemetryAPI.enabled` is `false`.

      - **settings.tracing.collector.opentelemetry.http**
        object

        Optional HTTP OTLP exporter settings (omit to use gRPC).

        - **settings.tracing.collector.opentelemetry.http.path**
          string

          OTLP traces HTTP path (defaults to `/v1/traces` in the chart when `http` is set but `path` is empty).

          Example:

          ```
          path: "/v1/traces"
          ```

        - **settings.tracing.collector.opentelemetry.http.timeout**
          string

          HTTP export timeout (Istio duration string).

          Pattern: `^([0-9]+(\.[0-9]+)?(ns|us|µs|ms|s|m|h))+$`

          Example:

          ```
          timeout: 10s
          ```

      - **settings.tracing.collector.opentelemetry.port**
        integer

        Collector port (for example `4317` for gRPC OTLP).

        Allowed values: `1 <= X <= 65535`

        Example:

        ```
        port: 4317
        ```

      - **settings.tracing.collector.opentelemetry.service**
        string

        OpenTelemetry collector service. The format is `[<namespace>/]<hostname>`.

        `<hostname>` must be the full hostname of a Service, including the cluster domain (for example, `opentelemetry-collector.observability.svc.cluster.local`), or a ServiceEntry host. Short names and IP addresses do not work. Specify `<namespace>` only if this hostname exists in several namespaces.

If Istio cannot find the service, spans are not exported.

Pattern: `^([a-z0-9]([-a-z0-9]{0,61}[a-z0-9])?/)?([a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?\.)*[a-zA-Z0-9]([-a-zA-Z0-9]{0,61}[a-zA-Z0-9])?$`

Example:

```
service: opentelemetry-collector.observability.svc.cluster.local
```

- **settings.tracing.collector.zipkin**
  object

  Zipkin protocol parameters used by Istio for sending traces. Jaeger accepts Zipkin ingestion on the standard port (`9411`).

  With tracing enabled, set `address` here for legacy mode or for Telemetry API when OpenTelemetry is not configured under `collector.opentelemetry`.

  - **settings.tracing.collector.zipkin.address**
    string

    Network address of the Zipkin collector in `<host>:<port>` format.

    If `telemetryAPI.enabled` is `true`, `<host>` must be the full hostname of a service known to Istio, including the cluster domain, for example `zipkin.myjaeger.svc.cluster.local` for a Kubernetes Service. Short names (such as `zipkin.myjaeger.svc`) and IP addresses do not work in this mode.

    Pattern: `^[0-9a-zA-Z.-]+:([1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-4][0-9]{3}|65[0-4][0-9]{2}|655[0-2][0-9]|6553[0-5])$`

    Example:

    ```
    address: zipkin.myjaeger.svc.cluster.local:9411
    ```

- **settings.tracing.enabled**
  boolean

  Turn on or off tracing collection and displaying in Kiali.

  Default: `false`

  Example:

  ```
  enabled: true
  ```

- **settings.tracing.kiali**
  object

  Span displaying settings for Kiali.

  When not configured, Kiali won’t show any tracing dashboards.

  Examples:

  ```
  kiali: {}
  ```

  ```
  kiali:
    jaegerURLForUsers: https://tracing-service:4443/jaeger
    jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
  ```

  - **settings.tracing.kiali.jaegerGRPCEndpoint**
    string

    Accessible from cluster address of jaeger GRPC interface for system queries by Kiali.

    When not configured, Kiali will only show external links using the `jaegerURLForUsers` config without interpretationing.

    Example:

    ```
    jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
    ```

  - **settings.tracing.kiali.jaegerURLForUsers**
    string

    Jaeger UI address for users. Mandatory parameter if Kiali is enabled.

    Example:

    ```
    jaegerURLForUsers: https://tracing-service:4443/jaeger
    ```

- **settings.tracing.sampling**
  number

  The sampling rate option can be used to control what percentage of requests get reported to your tracing system.

  This should be configured depending upon your traffic in the mesh and the amount of tracing data you want to collect.

  It is possible to override this option with the following Pod annotation:

  ```
  proxy.istio.io/config: |
    tracing:
      sampling: 100.0
  ```

  Default: `1.0`

  Allowed values: `0.01 <= X <= 100.0`

  Example:

  ```
  sampling: 50.05
  ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

IngressIstioController

Scope: Cluster
Version: v1alpha1

- **spec**
  object

  Required value

  - **spec.hostPort**
    object

    `HostPort` inlet settings.

    - **spec.hostPort.httpPort**
      integer

      Port for insecure HTTP connections.

      If the parameter is not set, the connection over HTTP cannot be established.

      This parameter is mandatory if `httpsPort` is not set.

      Example:

      ```
      httpPort: 80
      ```

    - **spec.hostPort.httpsPort**
      integer

      Port for secure HTTPS connections.

      If the parameter is not set, the connection over HTTPS cannot be established.

      This parameter is mandatory if `httpPort` is not set.

      Example:

      ```
      httpsPort: 443
      ```

  - **spec.ingressGatewayClass**
    string

    Required value

    Ingress gateway class is used by application Gateway resources for identifying the right Ingress gateway setup.

    The identification is organized by setting the spec.selector: `istio.deckhouse.io/ingress-gateway-class: <ingressGatewayClass value>`.

    This parameter does not change during the entire resource life.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Maximum length: `63`

Example:

```
ingressGatewayClass: istio
```

- **spec.inlet**
  string

  Required value

  The way traffic goes to cluster from the outer network.

  - `LoadBalancer` — Ingress controller is deployed and the service of `LoadBalancer` type is provisioned.
  - `NodePort` — Ingress controller is deployed and available through nodes' ports via `nodePort`.
  - `HostPort` — Ingress controller is deployed and available through nodes' ports via `hostPort`.

    > **Caution.** There can be only one controller with this inlet type on a host.

  Allowed values: `LoadBalancer`, `HostPort`, `NodePort`

- **spec.loadBalancer**
  object

  Not required value.

  A section of parameters of the `LoadBalancer` inlet.

  - **spec.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

  - **spec.loadBalancer.loadBalancerClass**
    string

    Class of the load balancer for incoming network requests (passed to the `spec.loadBalancerClass` parameter of the provisioned service with the `LoadBalancer` type).

- **spec.networkTopology**
  object

  Configures how the gateway extracts the client's original attributes (e.g. the source IP address) when it is deployed behind other proxies or load balancers.

  See Configuring Gateway Network Topology in the Istio documentation.

  - **spec.networkTopology.numTrustedProxies**
    integer

    Number of trusted proxies deployed in front of the Istio gateway proxy.

    It is used to correctly extract the client address from the `X-Forwarded-For` header and populate the `X-Envoy-External-Address` header consumed by upstream services.

    For example, if there is a cloud load balancer and a reverse proxy in front of the gateway, set this to `2`.

    It can be used with `proxyProtocol`. When both are configured and an incoming request contains an `X-Forwarded-For` header, Istio uses the trusted `X-Forwarded-For` chain in preference to the PROXY protocol attributes.

    Allowed values: `0 <= X`

    Example:

    ```
    numTrustedProxies: 2
    ```

  - **spec.networkTopology.proxyProtocol**
    boolean

    Enables the PROXY protocol on the gateway's TCP listeners, so that client attributes (such as

the source IP) forwarded by an upstream L4/TCP load balancer are preserved.

It can be used with `numTrustedProxies`. When both are configured and an incoming request contains an `X-Forwarded-For` header, Istio uses the trusted `X-Forwarded-For` chain in preference to the PROXY protocol attributes.

Example:

```
proxyProtocol: true
```

- **spec.nodePort**
  object

  `NodePort` inlet settings.

  - **spec.nodePort.httpPort**
    integer

    Port for insecure HTTP connections.

    If the parameter is not set, the connection over HTTP cannot be established.

    This parameter is mandatory if `httpsPort` is not set.

    Example:

    ```
    httpPort: 30080
    ```

  - **spec.nodePort.httpsPort**
    integer

    Port for secure HTTPS connections.

    If the parameter is not set, the connection over HTTPS cannot be established.

    This parameter is mandatory if `httpPort` is not set.

    Example:

    ```
    httpsPort: 30443
    ```

- **spec.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resourcesRequests**
  object

  Settings for CPU and memory requests and limits by ingress gateway pods.

  If omitted, the built-in VPA mode defaults are used. In `Static` mode, requests are specified directly in `static`. In `VPA` mode, the VPA adjusts requests within the configured minimum and maximum bounds.

  - **spec.resourcesRequests.mode**
    string

    Required value

    Resource management mode:

    - `Static` — allows you to specify requests directly in the `static` section;
    - `VPA` — uses VPA. You can configure this mode by modifying parameters in the `vpa` section.

    Default: `VPA`

    Allowed values: `VPA`, `Static`

  - **spec.resourcesRequests.static**
    object

    Resource management options for the `Static` mode.

    - **spec.resourcesRequests.static.cpu**

string

Configuring CPU requests.

Default: `100m`

- **spec.resourcesRequests.static.memory**
  string

  Configuring memory requests.

  Default: `128Mi`

- **spec.resourcesRequests.vpa**
  object

  Resource management options for the `VPA` mode.

  - **spec.resourcesRequests.vpa.cpu**
    object

    CPU-related VPA settings.

    - **spec.resourcesRequests.vpa.cpu.max**
      string

      The maximum value that the VPA can set for the CPU requests.

      Default: `1000m`

    - **spec.resourcesRequests.vpa.cpu.min**
      string

      The minimum value that the VPA can set for the CPU requests.

      Default: `100m`

  - **spec.resourcesRequests.vpa.memory**
    object

    Memory-related VPA settings.

    - **spec.resourcesRequests.vpa.memory.max**
      string

      The maximum memory requests the VPA can set.

      Default: `2048Mi`

    - **spec.resourcesRequests.vpa.memory.min**
      string

      The minimum memory requests the VPA can set.

      Default: `128Mi`

  - **spec.resourcesRequests.vpa.mode**
    string

    VPA operating mode.

    - `Initial` — VPA sets initial values for pod resource requests at pod creation time. Resource values are not changed automatically afterwards.
    - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod is recreated.
    - `Auto` — VPA automatically recreates pods to apply updated resource values. This mode is considered legacy starting from Deckhouse version 1.75; `InPlaceOrRecreate` is recommended.

    Default: `Initial`

    Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **spec.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard toleration list. Instance pods inherit this field as is.

        - **spec.tolerations.effect**
          string

          Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
        - **spec.tolerations.key**
          string
        - **spec.tolerations.operator**
          string

          Default: `Equal`

          Allowed values: `Exists`, `Equal`
        - **spec.tolerations.tolerationSeconds**
          integer
        - **spec.tolerations.value**
          string

IstioFederation

Scope: Cluster
Version: v1alpha1

Custom resource for setting remote cluster as trusted one.

- **spec**
  object

  Required value

  Available in editions: Ultimate/EE

  - **spec.metadata**
    object
    - **spec.metadata.ca**
      string

      Available in editions: Ultimate/EE

      Certificate for validation HTTPS endpoint with remote cluster metadata.

      Example:

      ```
      ca: "-----BEGIN CERTIFICATE----- ..... -----END CERTIFICATE-----"
      ```
    - **spec.metadata.insecureSkipVerify**
      boolean

      Available in editions: Ultimate/EE

      Skip validation certificate on HTTPS endpoint with remote cluster metadata.

      Default: `false`

      Example:

      ```
      insecureSkipVerify: true
      ```
  - **spec.metadataEndpoint**
    string

    Required value

    Available in editions: Ultimate/EE

    HTTPS endpoint with remote cluster metadata.

    Example:

    ```
    metadataEndpoint: https://istio.k8s.example.com/metadata/
    ```
  - **spec.trustDomain**
    string

    Required value

    Available in editions: Ultimate/EE

The TrustDomain of the remote cluster.

Deckhouse uses this value so that certificates from the remote cluster are trusted only for that remote domain, not for services in the local mesh.

Pattern: `^[0-9a-zA-Z._-]+$`

Example:

```
trustDomain: cluster.local
```

- **status**
  object
  - **status.conditions**
    array of objects

    Readiness of metadata exchange with the remote cluster.

    If `spec.trustDomain` equals the domain of this cluster, the resource does not take part in the federation: once the public metadata exchange succeeds, private metadata is not requested and `PrivateMetadataExchangeReady` is `False` with reason `ClusterDomainMismatch`; the remote cluster gets `UnknownSourceCluster` from this cluster.

    - `PublicMetadataExchangeReady` — remote `public.json` is reachable and valid. Reasons:
      - `Succeeded` (`True`) — the remote public metadata is received.
      - `FetchFailed` (`False`) — the request to the metadata endpoint failed: network, DNS, timeout, TLS, a 3xx redirect (redirects are not allowed), an address resolving to a restricted IP (loopback, link-local, unspecified, multicast), a response larger than 1 MiB. The error is in `message`.
      - `EndpointNotFound` (`False`) — HTTP 404: `metadataEndpoint` most likely does not point to the remote exporter, or the remote cluster has both federation and multicluster disabled.
      - `ServerError` (`False`) — HTTP 5xx: the remote exporter or its ingress is unavailable.
      - `AccessDenied` (`False`) — HTTP 401 or 403: something on the way to the remote exporter (an ingress or a proxy with access restrictions) refused the request; the code is in `message`.
      - `UnexpectedHTTPStatus` (`False`) — another non-200 HTTP status; the code is in `message`.
      - `InvalidJSON` (`False`) — the response is not valid JSON; check that `metadataEndpoint` points to the remote exporter.
      - `InvalidPublicMetadata` (`False`) — the remote `public.json` lacks `clusterUUID`, `authnKeyPub` or `rootCA`.
      - `ClusterUUIDMismatch` (`False`) — the UUID returned by the endpoint differs from `status.metadataCache.public.clusterUUID`: the remote cluster was recreated or `metadataEndpoint` now points to another cluster. Both UUIDs are in `message`.
    - `PrivateMetadataExchangeReady` — remote private metadata is reachable and valid. Reasons:
      - `AwaitingPublic` (`Unknown`) — the public metadata exchange is failing (see `PublicMetadataExchangeReady`), so private metadata is not requested.
      - `TokenGenerationFailed` (`False`) — the request could not be signed with the authn key of this cluster; check the key Secret in this cluster.
      - `FetchFailed` (`False`) — the request to the private endpoint failed: network, DNS, timeout, TLS, a 3xx redirect, a restricted IP, a response larger than 1 MiB. The error is in `message`.
      - `UnknownSourceCluster` (`False`) — the remote cluster does not know this cluster: it has no resource for this cluster, the public metadata exchange of that resource has never succeeded (`ClusterUUIDMismatch` there means this cluster was recreated, so the remote resource has to be recreated), or that resource is an `IstioFederation` whose `spec.trustDomain` equals the domain of the remote cluster (it shows `ClusterDomainMismatch` there).
      - `TrustLost` (`False`) — the same refusal as `UnknownSourceCluster`, but this cluster has already received private metadata from the remote one: the remote resource for this cluster was deleted, recreated or became ignored, or the authn key of this cluster changed. After a recreation or a key change the condition clears within a couple of minutes once the remote cluster fetches the public metadata of this cluster.
      - `WrongAllianceKind` (`False`) — the remote cluster registered this cluster as `IstioMulticluster`, not `IstioFederation`; replace it with an `IstioFederation` on the remote side.
      - `EndpointNotFound` (`False`) — HTTP 404: the public endpoint responded, so the remote cluster most likely has federation disabled (`federation.enabled`).
      - `ServerError` (`False`) — HTTP 5xx: the remote exporter failed on the private path.
      - `ClusterDomainMismatch` (`False`) — federation requires different cluster domains: either `spec.trustDomain` equals the domain of this cluster (private metadata is not requested; set `spec.trustDomain` to the domain of the remote cluster), or the remote cluster reports the domain of this cluster in its private metadata (not checked if the remote cluster does not publish its domain). The domain is in `message`.

        - `WrongJWTAudience` (`False`) — the remote cluster rejected the token as issued for another cluster: the public and private requests were served by different clusters (DNS, a load balancer or a proxy leads to several clusters).
        - `TokenExpired` (`False`) — the token has expired because the clocks differ; check time synchronization (NTP) on both clusters.
        - `InvalidToken` (`False`) — the remote cluster did not receive or could not parse the token: a proxy or ingress on the way strips the `Authorization` header, or a platform error.
        - `WrongJWTScope` (`False`) — the token was issued for another private path: a platform error or a proxy rewriting the path.
        - `AccessDenied` (`False`) — HTTP 401 or 403 without a machine-readable reason: the remote exporter is of an older version and does not tell why (most often it does not know this cluster), or an ingress or a proxy on the way refused the request; the code is in `message`.
        - `UnexpectedHTTPStatus` (`False`) — another non-200 HTTP status; the code is in `message`.
        - `InvalidJSON` (`False`) — the remote private response is not valid JSON.
        - `InvalidPrivateMetadata` (`False`) — the remote private metadata has no `publicServices`.
        - `MissingIngressGateways` (`False`) — the remote cluster published no ingress gateway addresses (usually its LoadBalancer has no address yet, or the remote exporter failed to collect them). The private metadata fetched last, including the gateway addresses, stays in use.
        - `Succeeded` (`True`) — the remote private metadata is received.
      - `DataplaneConnectionReady` — cross-cluster dataplane health probe to the remote `alliance-healthcheck` service succeeded. Reasons:
        - `AwaitingPublic` (`Unknown`) — `status.metadataCache` has no public metadata yet.
        - `AwaitingPrivate` (`Unknown`) — `status.metadataCache` has public metadata but no private metadata yet. For a new connection the state is kept for one or two probe cycles (up to about two minutes) after the private metadata is received, so that the mesh applies it before the first probe.
        - `ProbeFailed` (`False`) — the probe through the remote ingress gateway fails; `message` has the request error or the HTTP status code. With a remote cluster on an older module version (it neither checks the resource kind nor publishes its domain), a wrong resource kind on the remote side or equal cluster domains with `spec.trustDomain` set to another domain show up only here.
        - `ConfigurationError` (`False`) — the private metadata is received, but the remote `publicServices` has no `alliance-healthcheck` service (for example, the remote istio module version is older). Details are in `message`.
        - `MetadataUnavailable` (`False`) — platform error: the remote cluster UUID cannot be read from the cached metadata.
        - `Succeeded` (`True`) — the dataplane connection to the remote cluster works.
      - **status.conditions.lastProbeTime**
        string

        Required value
      - **status.conditions.lastTransitionTime**
        string

        Required value
      - **status.conditions.message**
        string

        Required value
      - **status.conditions.reason**
        string

        Required value
      - **status.conditions.status**
        string

        Required value

        Allowed values: `True`, `False`, `Unknown`
      - **status.conditions.type**
        string

        Required value
  - **status.metadataCache**
    object
    - **status.metadataCache.private**
      object

        - **status.metadataCache.private.clusterDomain**
          string
        - **status.metadataCache.private.ingressGateways**
          array of objects
          - **status.metadataCache.private.ingressGateways.address**
            string
          - **status.metadataCache.private.ingressGateways.port**
            integer
        - **status.metadataCache.private.publicServices**
          array of objects
          - **status.metadataCache.private.publicServices.hostname**
            string
          - **status.metadataCache.private.publicServices.ports**
            array of objects
            - **status.metadataCache.private.publicServices.ports.name**
              string
            - **status.metadataCache.private.publicServices.ports.port**
              integer
            - **status.metadataCache.private.publicServices.ports.protocol**
              string
      - **status.metadataCache.public**
        object
        - **status.metadataCache.public.authnKeyPub**
          string
        - **status.metadataCache.public.clusterUUID**
          string
        - **status.metadataCache.public.rootCA**
          string

IstioMulticluster

Scope: Cluster
Version: v1alpha1

Custom resource for setting remote cluster as trusted one.

- **spec**
  object

  Required value

  Available in editions: Ultimate/EE

  - **spec.enableIngressGateway**
    boolean

    Available in editions: Ultimate/EE

    Whether to reach the pods of the remote cluster through its ingress gateway.

    If the pods of the clusters are directly reachable from each other (flat network), set it to `false` to avoid the extra hop through the ingress gateway.

    Set the same value on both sides: in the remote cluster, in the IstioMulticluster that describes this cluster. A cluster deploys its ingress gateway only if federation is enabled or at least one IstioMulticluster in it has `enableIngressGateway: true`. If `false` is set on one side only, the other side may find no ingress gateway to connect through (the `IngressGatewayNotDeployed` reason in `status.conditions`).

    Default: `true`

  - **spec.metadata**
    object
    - **spec.metadata.ca**
      string

      Available in editions: Ultimate/EE

      HTTPS certificate authority for remote cluster metadata.

    - **spec.metadata.insecureSkipVerify**
      boolean

      Available in editions: Ultimate/EE

      HTTPS check or not for remote cluster metadata.

      Default: `false`

Example:

```
insecureSkipVerify: true
```

- **spec.metadataEndpoint**
  string

  Required value

  Available in editions: Ultimate/EE

  HTTPS endpoint with remote cluster metadata.

  Example:

  ```
  metadataEndpoint: https://istio.k8s.example.com/metadata/
  ```

- **status**
  object
  - **status.conditions**
    array of objects

    Readiness of metadata exchange with the remote cluster.

    - `PublicMetadataExchangeReady` — remote `public.json` is reachable and valid. Reasons:
      - `Succeeded` (`True`) — the remote public metadata is received.
      - `FetchFailed` (`False`) — the request to the metadata endpoint failed: network, DNS, timeout, TLS, a 3xx redirect (redirects are not allowed), an address resolving to a restricted IP (loopback, link-local, unspecified, multicast), a response larger than 1 MiB. The error is in `message`.
      - `EndpointNotFound` (`False`) — HTTP 404: `metadataEndpoint` most likely does not point to the remote exporter, or the remote cluster has both federation and multicluster disabled.
      - `ServerError` (`False`) — HTTP 5xx: the remote exporter or its ingress is unavailable.
      - `AccessDenied` (`False`) — HTTP 401 or 403: something on the way to the remote exporter (an ingress or a proxy with access restrictions) refused the request; the code is in `message`.
      - `UnexpectedHTTPStatus` (`False`) — another non-200 HTTP status; the code is in `message`.
      - `InvalidJSON` (`False`) — the response is not valid JSON; check that `metadataEndpoint` points to the remote exporter.
      - `InvalidPublicMetadata` (`False`) — the remote `public.json` lacks `clusterUUID`, `authnKeyPub` or `rootCA`.
      - `ClusterUUIDMismatch` (`False`) — the UUID returned by the endpoint differs from `status.metadataCache.public.clusterUUID`: the remote cluster was recreated or `metadataEndpoint` now points to another cluster. Both UUIDs are in `message`.
    - `PrivateMetadataExchangeReady` — remote private metadata is reachable and valid. Reasons:
      - `AwaitingPublic` (`Unknown`) — the public metadata exchange is failing (see `PublicMetadataExchangeReady`), so private metadata is not requested.
      - `TokenGenerationFailed` (`False`) — the request could not be signed with the authn key of this cluster; check the key Secret in this cluster.
      - `FetchFailed` (`False`) — the request to the private endpoint failed: network, DNS, timeout, TLS, a 3xx redirect, a restricted IP, a response larger than 1 MiB. The error is in `message`.
      - `UnknownSourceCluster` (`False`) — the remote cluster does not know this cluster: it has no resource for this cluster, the public metadata exchange of that resource has never succeeded (`ClusterUUIDMismatch` there means this cluster was recreated, so the remote resource has to be recreated), or the remote cluster has an `IstioFederation` for this cluster instead of an `IstioMulticluster` (with equal domains, as multicluster requires, it is ignored there, so the wrong kind shows up here and not as `WrongAllianceKind`).
      - `TrustLost` (`False`) — the same refusal as `UnknownSourceCluster`, but this cluster has already received private metadata from the remote one: the remote resource for this cluster was deleted, recreated or became ignored, or the authn key of this cluster changed. After a recreation or a key change the condition clears within a couple of minutes once the remote cluster fetches the public metadata of this cluster.
      - `WrongAllianceKind` (`False`) — the remote cluster registered this cluster as `IstioFederation`, not `IstioMulticluster`; replace it with an `IstioMulticluster` on the remote side.
      - `EndpointNotFound` (`False`) — HTTP 404: the public endpoint responded, so the remote cluster most likely has multicluster disabled (`multicluster.enabled`).
      - `ServerError` (`False`) — HTTP 5xx: the remote exporter failed on the private path.
      - `ClusterDomainMismatch` (`False`) — the remote cluster domain differs from the domain of this cluster, while multicluster requires equal domains. Both domains are in `message`. Not checked if the remote cluster does not publish its domain.

    - `WrongJWTAudience` (`False`) — the remote cluster rejected the token as issued for another cluster: the public and private requests were served by different clusters (DNS, a load balancer or a proxy leads to several clusters).
    - `TokenExpired` (`False`) — the token has expired because the clocks differ; check time synchronization (NTP) on both clusters.
    - `InvalidToken` (`False`) — the remote cluster did not receive or could not parse the token: a proxy or ingress on the way strips the `Authorization` header, or a platform error.
    - `WrongJWTScope` (`False`) — the token was issued for another private path: a platform error or a proxy rewriting the path.
    - `AccessDenied` (`False`) — HTTP 401 or 403 without a machine-readable reason: the remote exporter is of an older version and does not tell why (most often it does not know this cluster), or an ingress or a proxy on the way refused the request; the code is in `message`.
    - `UnexpectedHTTPStatus` (`False`) — another non-200 HTTP status; the code is in `message`.
    - `InvalidJSON` (`False`) — the remote private response is not valid JSON.
    - `InvalidPrivateMetadata` (`False`) — the remote private metadata has an empty `networkName` or `apiHost`.
    - `MissingIngressGateways` (`False`) — `spec.enableIngressGateway` is `true`, but the remote cluster published no ingress gateway addresses (usually its LoadBalancer has no address yet, or the remote exporter failed to collect them). A remote cluster on an older module version that deploys no ingress gateway is also reported with this reason, not with `IngressGatewayNotDeployed`. The private metadata fetched last, including the gateway addresses, stays in use.
    - `IngressGatewayNotDeployed` (`False`) — `spec.enableIngressGateway` is `true`, but the remote cluster deploys no ingress gateway. Most often, the IstioMulticluster that describes this cluster in the remote cluster has `spec.enableIngressGateway: false`, and nothing else there needs the ingress gateway (see `spec.enableIngressGateway`); `message` lists what else to check in the remote cluster. The cached addresses are cleared, and this cluster does not connect to the remote cluster until this is fixed.
    - `Succeeded` (`True`) — the remote private metadata is received.
  - `RemoteAPIServerReady` — remote multicluster API host (`apiHost`) serves the authenticated `/api` endpoint. Reasons:
    - `AwaitingPrivate` (`Unknown`) — the private metadata exchange has not succeeded yet (see `PrivateMetadataExchangeReady`), including while the public metadata exchange fails.
    - `TokenGenerationFailed` (`False`) — the API token could not be signed with the authn key of this cluster; check the key Secret in this cluster.
    - `MissingAPIHost` (`False`) — the remote `apiHost` consists only of whitespace (an empty `apiHost` is reported as `InvalidPrivateMetadata` on `PrivateMetadataExchangeReady`).
    - `RemoteAPIUnreachable` (`False`) — the request to the remote API failed: network, DNS, timeout, TLS, a redirect, a non-HTTPS `apiHost`, a restricted IP, a response larger than 1 MiB. The error is in `message`.
    - `RemoteAPIBadResponse` (`False`) — the remote API returned a non-200 HTTP status (the code is in `message`); check the remote api-proxy and its permissions.
    - `RemoteAPIInvalidResponse` (`False`) — the remote API response is not valid JSON; check that `apiHost` points to the remote api-proxy.
    - `RemoteAPIUnexpectedResponse` (`False`) — `apiHost` does not serve the Kubernetes API (the response kind is not `APIVersions`).
    - `RemoteAPIReachable` (`True`) — the remote API server is reachable.
  - `DataplaneConnectionReady` — cross-cluster dataplane health probe to the remote `alliance-healthcheck` service succeeded. Reasons:
    - `AwaitingPublic` (`Unknown`) — `status.metadataCache` has no public metadata yet.
    - `AwaitingPrivate` (`Unknown`) — `status.metadataCache` has public metadata but no private metadata yet. For a new connection the state is kept for one or two probe cycles (up to about two minutes) after the private metadata is received, so that the mesh applies it before the first probe.
    - `ProbeFailed` (`False`) — the probe to the remote `alliance-healthcheck` service fails (through the remote ingress gateway, or directly to remote Pods when `spec.enableIngressGateway` is `false`); `message` has the request error or the HTTP status code. With a remote cluster on an older module version, a wrong resource kind or different cluster domains show up only here.
    - `MetadataUnavailable` (`False`) — platform error: the remote cluster UUID cannot be read from the cached metadata.
    - `Succeeded` (`True`) — the dataplane connection to the remote cluster works.
  - **status.conditions.lastProbeTime**
    string

    Required value

  - **status.conditions.lastTransitionTime**
    string

    Required value

        - **status.conditions.message**
          string

          Required value
        - **status.conditions.reason**
          string

          Required value
        - **status.conditions.status**
          string

          Required value

          Allowed values: `True`, `False`, `Unknown`
        - **status.conditions.type**
          string

          Required value
    - **status.metadataCache**
      object
      - **status.metadataCache.private**
        object
        - **status.metadataCache.private.ambientGateways**
          array of objects
          - **status.metadataCache.private.ambientGateways.address**
            string
          - **status.metadataCache.private.ambientGateways.port**
            integer
        - **status.metadataCache.private.apiHost**
          string
        - **status.metadataCache.private.ca**
          string
        - **status.metadataCache.private.clusterDomain**
          string
        - **status.metadataCache.private.clusterID**
          string
        - **status.metadataCache.private.ingressGateways**
          array of objects
          - **status.metadataCache.private.ingressGateways.address**
            string
          - **status.metadataCache.private.ingressGateways.port**
            integer
        - **status.metadataCache.private.networkName**
          string
      - **status.metadataCache.public**
        object
        - **status.metadataCache.public.authnKeyPub**
          string
        - **status.metadataCache.public.clusterUUID**
          string
        - **status.metadataCache.public.rootCA**
          string

WaypointInstance

Short names: wpi

Scope: Namespaced
Version: v1alpha1

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `51`
- **spec**
  object
  - **spec.allowedRoutes**
    object

    Configuration for restricting which Routes can attach to the Gateway.

- **spec.allowedRoutes.namespaces**
  object

  Configuration for restricting the namespaces of Routes that can attach to the Gateway.

  - **spec.allowedRoutes.namespaces.from**
    string

    Specifies which namespaces are allowed to attach Routes to this Gateway.

    Possible values:

    - `All`: Routes in all namespaces may be attached.
    - `Selector`: Only Routes in namespaces matching the `selector` may be attached.
    - `Same`: Only Routes in the same namespace as the Gateway may be attached.

    Default: `Same`

    Allowed values: `All`, `Selector`, `Same`

  - **spec.allowedRoutes.namespaces.selector**
    object

    Label selector for selecting namespaces.

    Required when `from` is `Selector`.

    - **spec.allowedRoutes.namespaces.selector.matchExpressions**
      array of objects

      A list of label selector requirements.

      - **spec.allowedRoutes.namespaces.selector.matchExpressions.key**
        string

        Required value

      - **spec.allowedRoutes.namespaces.selector.matchExpressions.operator**
        string

        Required value

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.allowedRoutes.namespaces.selector.matchExpressions.values**
        array of strings
    - **spec.allowedRoutes.namespaces.selector.matchLabels**
      object

      A map of {key,value} pairs.

      A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions with operator `In` and the values array containing only the value.

- **spec.nodeSelector**
  object

  Node selector for waypoint proxy pods.

- **spec.replicasManagement**
  object

  Replication management settings and scaling for the waypoint instance.

  - **spec.replicasManagement.hpa**
    object

    Options for replicas management for the `HPA` mode.

    - **spec.replicasManagement.hpa.maxReplicas**
      integer

      The upper limit for the number of replicas to which the HPA can scale up.

      Allowed values: `1 <= X`

    - **spec.replicasManagement.hpa.metrics**
      array of objects

      The HPA will use these metrics to decide whether to increase or decrease the number of replicas.

- **spec.replicasManagement.hpa.metrics.targetAverageUtilization**
  integer

  Required value

  The target value of the average resource metric across all relevant pods, represented as a percentage of the requested value.

  Allowed values: `1 <= X <= 100`

- **spec.replicasManagement.hpa.metrics.type**
  string

  Required value

  Metric type.

  Allowed values: `CPU`

- **spec.replicasManagement.hpa.minReplicas**
  integer

  The lower limit for the number of replicas to which the HPA can scale down.

  Allowed values: `1 <= X`

- **spec.replicasManagement.mode**
  string

  Replicas management mode.

  Possible values:

  - `Static`: the mode where the number of replicas is specified explicitly.
  - `HPA`: the mode where the number of replicas is calculated automatically using HPA based on CPU usage.

  Default: `Static`

  Allowed values: `Static`, `HPA`

- **spec.replicasManagement.static**
  object

  Options for replicas management for the `Static` mode.

  - **spec.replicasManagement.static.replicas**
    integer

    Desired number of replicas.

    Allowed values: `1 <= X`

- **spec.resourcesManagement**
  object

  CPU and memory request and limit settings for the waypoint Pods.

  - **spec.resourcesManagement.mode**
    string

    Resource management mode.

    Default: `VPA`

    Allowed values: `VPA`, `Static`

  - **spec.resourcesManagement.static**
    object

    Static mode settings.

    - **spec.resourcesManagement.static.limits**
      object

      Resource limits.

      - **spec.resourcesManagement.static.limits.cpu**
        string

        CPU limits.

      - **spec.resourcesManagement.static.limits.memory**

string

Memory limits.

- **spec.resourcesManagement.static.requests**
  object

  Resource requests.

  - **spec.resourcesManagement.static.requests.cpu**
    string

    CPU requests.

    Default: `100m`

  - **spec.resourcesManagement.static.requests.memory**
    string

    Memory requests.

    Default: `128Mi`

- **spec.resourcesManagement.vpa**
  object

  Resource management settings for the VPA mode.

  - **spec.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **spec.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      If set, the limits are calculated based on the requests and the specified ratio.

    - **spec.resourcesManagement.vpa.cpu.max**
      string

      Maximum value of allowed CPU requests to be submitted by the VPA.

      Default: `1000m`

    - **spec.resourcesManagement.vpa.cpu.min**
      string

      Minimum value of allowed CPU requests to be submitted by the VPA.

      Default: `100m`

  - **spec.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **spec.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      If set, the limits are calculated based on the requests and the specified ratio.

    - **spec.resourcesManagement.vpa.memory.max**
      string

      Maximum value of allowed memory requests to be submitted by the VPA.

      Default: `2048Mi`

    - **spec.resourcesManagement.vpa.memory.min**
      string

      Minimum value of allowed memory requests to be submitted by the VPA.

      Default: `128Mi`

  - **spec.resourcesManagement.vpa.mode**
    string

VPA operating mode.

Default: `InPlaceOrRecreate`

Allowed values: `Initial`, `InPlaceOrRecreate`

- **spec.tolerations**
  array of objects

  Tolerations for waypoint proxy pods.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

    Allowed values: `Exists`, `Equal`
  - **spec.tolerations.tolerationSeconds**
    integer
  - **spec.tolerations.value**
    string
- **spec.waypointFor**
  string

  Controls the `istio.io/waypoint-for` label on all managed resources.

  Default: `All`

  Allowed values: `All`, `Service`, `Workload`

- **status**
  object
  - **status.observedGeneration**
    integer

    Allowed values: `0 <= X`
  - **status.synced**
    boolean

    True when desired and actual resources are in sync.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources (by istio.io)

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

Routing

DestinationRule

Reference.

Allows you to:

- Define a strategy for balancing traffic between service endpoints:
  - The balancing algorithm to use (LEAST_CONN, ROUND_ROBIN, ...);
  - Symptoms that an endpoint is unhealthy and rules for removing it from load balancing;
  - Limits for TCP connections and endpoint requests;
  - Sticky Sessions;
  - Circuit Breaker;
- Define alternative endpoint groups for traffic processing (suitable for Canary Deployments). Also, you can configure balancing strategies for each group;
- Configuring TLS for outgoing requests.

VirtualService

Reference.

Using VirtualService is optional; regular services fit just fine if their capabilities match your requirements.

Allows you to configure request routing:

- Arguments on which routing decisions are based on:
  - Host;
  - URI;
  - Weight;
- Parameters of the resulting directions:
  - The new host;
  - The new URI;
  - If the host is defined using DestinationRule, then requests can be sent to `subsets`;
  - Timeout and retry settings.

⚠ Istio must be aware of the `destination`; if you use an external API, register it via ServiceEntry.

ServiceEntry

Reference.

It is similar to Endpoints + Service in vanilla Kubernetes. Informs Istio about the existence of an external service and lets you redefine its address.

Authentication

Answers the question “Who has made the request?” Not to be confused with authorization - the latter determines what the authenticated subject can or cannot do.

There are two authentication methods:

- mTLS;
- JWT tokens;

PeerAuthentication

Reference.

Allows you to define the mTLS strategy for an individual namespace. Defines how traffic will be tunneled (or not) to the sidecar. Each mTLS request can automatically identify the source and allows you to use it in the authorization rules.

RequestAuthentication

Reference.

Allows you to configure JWT authentication for requests.

Authorization

⚠ Authorization without the use of mTLS or JWT authentication will not work fully. In this case, you will be able to use only basic arguments, such as `source.ip` and `request.headers`, for defining policies.

AuthorizationPolicy

Reference.

Enables and defines access control to the `workload`. The `AuthorizationPolicy` custom resource supports both ALLOW and DENY rules. The following decision-making algorithm is used if at least one policy is defined for a workload:

- The request is denied if it falls under the DENY policy.
- The request is allowed if there are no ALLOW policies for the service.
- The request is allowed if it falls under the ALLOW policy.
- In all other cases, the request is denied.

The following arguments are used in the decision-making algorithm:

- source:
  - namespace
  - principal (i.e., the ID the user has received after authentication)
  - IP
- destination:
  - method (GET, POST, ...)
  - Host
  - port
  - URI
- conditions:
  - HTTP-headers
  - source arguments
  - destination arguments
  - JWT-tokens

Sidecar

Reference

This resource limits the number of services for which information is transmitted to the `istio-proxy` sidecar.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> **i** The module lifecycle stage: General Availability
>
> The module has requirements for installation

Circuit Breaker

The `outlierDetection` settings in the DestinationRule custom resource help to determine whether some endpoints do not behave as expected. Refer to the Envoy documentation for more details on the Outlier Detection algorithm.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: reviews-cb-policy
spec:
  host: reviews.prod.svc.cluster.local
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100 # The maximum number of connections to the host (cumulative
for all endpoints)
      http:
        maxRequestsPerConnection: 10 # The connection will be re-established after every
10 requests
    outlierDetection:
      consecutive5xxErrors: 7 # Seven consecutive errors are allowed (including 5XX, TCP
and HTTP timeouts)
      interval: 5m          # over 5 minutes.
      baseEjectionTime: 15m   # Upon reaching the error limit, the endpoint will be
excluded from balancing for 15 minutes.
```

Additionally, the VirtualService resource is used to configure the HTTP timeouts. These timeouts are also taken into account when calculating error statistics for endpoints.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: my-productpage-rule
  namespace: myns
spec:
  hosts:
  - productpage
  http:
  - timeout: 5s
    route:
    - destination:
        host: productpage
```

gRPC balancing

⚠ Assign a name with the `grpc` prefix or value to the port in the corresponding service to make gRPC service balancing start automatically.

Locality Failover

i Read the Istio Locality Failover documentation if needed.

Istio allows you to configure a priority-based locality (geographic location) failover between endpoints. Istio uses node labels with the appropriate hierarchy to define the zone:

- `topology.istio.io/subzone`
- `topology.kubernetes.io/zone`
- `topology.kubernetes.io/region`

This comes in handy for inter-cluster failover when used together with a multicluster.

⚠ The Locality Failover can be enabled using the DestinationRule CR. Note that you also have to configure the outlierDetection.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
spec:
  host: helloworld
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true # Locality Failover is enabled
    outlierDetection: # outlierDetection must be enabled
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m
```

Retry

You can use the VirtualService resource to configure Retry for requests.

> ⚠ All requests (including POST ones) are retried three times by default.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: ratings-route
spec:
  hosts:
  - ratings.prod.svc.cluster.local
  http:
  - route:
    - destination:
        host: ratings.prod.svc.cluster.local
    retries:
      attempts: 3
      perTryTimeout: 2s
      retryOn: gateway-error,connect-failure,refused-stream
```

Canary

Istio is only responsible for flexible request routing that relies on special request headers (such as cookies) or simply randomness. The CI/CD system is responsible for customizing this routing and switching between canary versions.

The idea is that two Deployments with different versions of the application are deployed in the same namespace. The Pods of different versions have different labels (`version: v1` and `version: v2`).

You have to configure two custom resources:

- A DestinationRule – defines how to identify different versions of your application (subsets);
- A VirtualService – defines how to balance traffic between different versions of your application.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: productpage-canary
spec:
  host: productpage
  # subsets are only available when accessing the host via the VirtualService from a Pod
managed by Istio.
  # These subsets must be defined in the routes.
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

Cookie-based routing

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - match:
    - headers:
       cookie:
         regex: "^(.*;?)?(canary=yes)(;.*)?"
    route:
    - destination:
        host: productpage
        subset: v2 # The reference to the subset from the DestinationRule.
  - route:
    - destination:
        host: productpage
        subset: v1
```

Probability-based routing

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1 # The reference to the subset from the DestinationRule.
      weight: 90 # Percentage of traffic that the Pods with the version: v1 label will be
getting.
  - route:
    - destination:
        host: productpage
        subset: v2
      weight: 10
```

Ingress to publish applications

Istio Ingress Gateway

The IngressIstioController custom resource spins up a dedicated Istio Ingress Gateway proxy. Each controller instance gets its own gateway class, and you select the instance from an Istio `Gateway` resource by referencing the matching `istio.deckhouse.io/ingress-gateway-class` label. The module manages the gateway workload and its `Service`, while the `Gateway` and routing resources (`VirtualService`) remain yours to manage.

Start by creating an `IngressIstioController`. In the example below, HTTP and HTTPS are exposed on the selected frontend nodes using host ports:

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  # The value selected by Gateway resources via the istio.deckhouse.io/ingress-gateway-
class label.
  ingressGatewayClass: istio-hp
  # IngressIstioController works with LoadBalancer, NodePort, and HostPort inlets.
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  resourcesRequests:
    mode: VPA
```

Note that the TLS secret for an ingress gateway must be created in the `d8-ingress-istio` namespace, not in your application’s namespace — this is an easy detail to miss.

```
apiVersion: v1
kind: Secret
metadata:
  name: app-tls-secret
  namespace: d8-ingress-istio # note the namespace isn't app-ns
type: kubernetes.io/tls
data:
  tls.crt: |
    <tls.crt data>
  tls.key: |
    <tls.key data>
```

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway-app
  namespace: app-ns
spec:
  selector:
    # label selector for using the Istio Ingress Gateway main-hp
    istio.deckhouse.io/ingress-gateway-class: istio-hp
  servers:
    - port:
        # standard template for using the HTTP protocol
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - app.example.com
    - port:
        # standard template for using the HTTPS protocol
        number: 443
        name: https
        protocol: HTTPS
      tls:
        mode: SIMPLE
        # a secret with a certificate and a key, which must be created in the d8-ingress-
istio namespace
        # supported secret formats can be found at
https://istio.io/latest/docs/tasks/traffic-management/ingress/secure-ingress/#key-
formats
        credentialName: app-tls-secret
      hosts:
        - app.example.com
```

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            host: app-svc
```

For the full list of controller settings — load-balancer annotations, network topology, scheduling, and resource management — see the IngressIstioController custom resource reference.

Preserving client attributes behind external proxies

When the gateway is deployed behind other proxies or load balancers (for example, a cloud load balancer or a reverse proxy), configure `spec.networkTopology` so that the gateway can correctly extract the client's original attributes, such as the source IP address. See Configuring Gateway Network Topology in the Istio documentation for details.

Use `numTrustedProxies` when the upstream proxies pass the client IP address in the `X-Forwarded-For` header. Set it to the number of trusted proxies deployed in front of the gateway so that Istio extracts the correct client address and populates the `X-Envoy-External-Address` header for upstream services. For example, if a cloud load balancer and a reverse proxy sit in front of the gateway, set the value to `2`:

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  ingressGatewayClass: istio-hp
  inlet: LoadBalancer
  networkTopology:
    numTrustedProxies: 2
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  resourcesRequests:
    mode: VPA
```

Use `proxyProtocol` when an upstream L4/TCP load balancer forwards the client attributes via the PROXY protocol instead of HTTP headers. When this parameter is enabled, the gateway starts parsing the PROXY protocol header on incoming TCP connections:

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  ingressGatewayClass: istio-hp
  inlet: LoadBalancer
  networkTopology:
    proxyProtocol: true
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  resourcesRequests:
    mode: VPA
```

i `numTrustedProxies` and `proxyProtocol` can be used together. When both are configured and an incoming request contains an `X-Forwarded-For` header, Istio uses the trusted `X-Forwarded-For` chain in preference to the PROXY protocol attributes.

Managing gateway resource requests

Use `spec.resourcesRequests` to control CPU and memory requests for the ingress gateway pods. Two modes are available:

- `Static` — requests are specified directly and stay fixed:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: IngressIstioController
  metadata:
    name: main
  spec:
    ingressGatewayClass: istio-hp
    inlet: HostPort
    hostPort:
      httpPort: 80
      httpsPort: 443
    resourcesRequests:
      mode: Static
      static:
        cpu: 100m
        memory: 128Mi
  ```

- `VPA` — a Vertical Pod Autoscaler adjusts requests within the configured `min`/`max` bounds. Starting from DP version 1.75, the recommended VPA mode is `InPlaceOrRecreate`, which updates pod resources in place when the cluster supports it and falls back to recreating the pod otherwise (the

legacy `Auto` mode always recreates the pod):

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  ingressGatewayClass: istio-hp
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  resourcesRequests:
    mode: VPA
    vpa:
      mode: InPlaceOrRecreate
      cpu:
        min: 100m
        max: 1000m
      memory:
        min: 128Mi
        max: 2048Mi
```

Ingress NGINX

To use Ingress, you need to:

- Configure the Ingress controller by adding Istio sidecar to it. In our case, you need to enable the `enableIstioSidecar` parameter in the ingress-nginx module's IngressNginxController custom resource.
- Set up an Ingress that refers to the Service. The following annotations are mandatory for Ingress:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — using this annotation, the Ingress controller sends requests to a single ClusterIP (from Service CIDR) while envoy load balances them. Ingress controller's sidecar is only catching traffic directed to Service CIDR.
  - `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — using this annotation, the sidecar container can identify the application service that serves requests.

Examples:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Nginx proxies traffic to the ClusterIP instead of pods' own IPs.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # In Istio, all routing is carried out based on the `Host:` headers.
    # Instead of letting Istio know about the `productpage.example.com` external domain,
    # we use the internal domain of which Istio is aware.
    nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc
spec:
  rules:
    - host: productpage.example.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: productpage
              port:
                number: 9080
```

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
  - name: http
    port: 9080
  selector:
    app: productpage
  type: ClusterIP
```

Authorization configuration examples

Decision-making algorithm

The following algorithm for deciding the fate of a request becomes active after `AuthorizationPolicy` is created for the application:

- The request is denied if it falls under the DENY policy;
- The request is allowed if there are no ALLOW policies for the application;
- The request is allowed if it falls under the ALLOW policy.
- All other requests are denied.

In other words, if you explicitly deny something, then only this restrictive rule will work. If you explicitly allow something, only explicitly authorized requests will be allowed (however, restrictions will stay in force and have precedence).

⚠ The policies based on high-level parameters like namespace or principal require enabling Istio for all involved applications. Also, there must be organized Mutual TLS between applications.

Examples:

- Let’s deny POST requests for the myapp application. Since a policy is defined, only POST requests to the application are denied (as per the algorithm above).

  ```
  apiVersion: security.istio.io/v1beta1
  kind: AuthorizationPolicy
  metadata:
    name: deny-post-requests
    namespace: foo
  spec:
    selector:
      matchLabels:
        app: myapp
    action: DENY
    rules:
    - to:
      - operation:
          methods: ["POST"]
  ```

- Below, the ALLOW policy is defined for the application. It only allows requests from the `bar` namespace (other requests are denied).

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # The default value, can be skipped.
  rules:
  - from:
    - source:
       namespaces: ["bar"]
```

- Below, the ALLOW policy is defined for the application. Note that it does not have any rules, so not a single request matches it (still, the policy exists). Thus, our decision-making algorithm suggests that if something is allowed, then everything else is denied. In this case, “everything else” includes all the requests.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # The default value, can be skipped.
  rules: []
```

- Below, the (default) ALLOW policy is defined for the application. Note that it has an empty rule. Any request matches this rule, so it is naturally approved.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  rules:
  - {}
```

Deny all actions for the foo namespace

There are two ways you can do that:

- Explicitly. Here, the DENY policy is created. It has a single `{}` rule that covers all the requests:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  action: DENY
  rules:
  - {}
```

- Implicitly. Here, the (default) ALLOW policy is created that does not have any rules. Thus, no requests

will match it, and the policy will deny all of them.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec: {}
```

Deny requests from the foo NS only

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: deny-from-ns-foo
 namespace: myns
spec:
 action: DENY
 rules:
 - from:
   - source:
      namespaces: ["foo"]
```

Allow requests for the foo NS only

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-intra-namespace-only
 namespace: foo
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      namespaces: ["foo"]
```

Allow requests from anywhere in the cluster

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-my-cluster
 namespace: myns
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      principals: ["mycluster.local/*"]
```

Allow any requests for foo or bar clusters

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-foo-or-bar-clusters-to-ns-baz
 namespace: baz
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      principals: ["foo.local/*", "bar.local/*"]
```

Allow any requests only from entities in the baz namespace of the foo or bar clusters

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-foo-or-bar-clusters-to-ns-baz
 namespace: baz
spec:
 action: ALLOW
 rules:
 - from:
   - source: # Logical conjunction is used for the rules below.
      namespaces: ["baz"]
      principals: ["foo.local/*", "bar.local/*"]
```

Allow from any cluster (via mTLS)

⚠ The denying rules (if they exist) have priority over any other rules. For details, refer to Decision-making algorithm.

Example:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-any-cluster-with-mtls
 namespace: myns
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      principals: ["*"] # To force the mTLS usage.
```

Allow all requests from anywhere (including no mTLS - plain text traffic)

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-any
 namespace: myns
spec:
 action: ALLOW
 rules: [{}]
```

Setting up federation for two clusters using the IstioFederation CR

⚠ Federation covers sidecar-mode workloads only. For details, refer to Ambient mesh limitations.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Setting up multicluster for two clusters using the IstioMulticluster CR

⚠ Multicluster covers sidecar-mode workloads only. For details, refer to Ambient mesh limitations.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Ambient mesh

⚠ Ambient mesh support is experimental and not recommended for production use.

⚠ Ambient-mode workloads cannot take part in federation or multicluster. For details, refer to Ambient mesh limitations.

The ambient mesh components mentioned in this section are described on the module overview page.

Enabling ambient mesh

Ambient mode requires the `CNIPlugin` traffic redirection mode and Istio 1.25 or newer.

The following is a module configuration example with the ambient mode enabled:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  enabled: true
  version: 2
  settings:
    dataPlane:
      trafficRedirectionSetupMode: CNIPlugin
    ambient:
      enabled: true
```

Once enabled, the module runs the `ztunnel` DaemonSet and the waypoint controller. To enroll workloads into the ambient mesh, follow the steps below.

Enrolling workloads into the ambient (L4) mesh

Add the `istio.io/dataplane-mode=ambient` label to a namespace to capture the traffic of its pods with `ztunnel`. This provides L4 features (mutual TLS, identity, L4 authorization) without a sidecar:

```
d8 k label namespace myns istio.io/dataplane-mode=ambient
```

Adding a waypoint for L7 features

To get L7 features (HTTP routing, L7 authorization, richer telemetry), create a WaypointInstance resource in the namespace:

The following is a WaypointInstance resource example, which creates a waypoint for all workloads and services in a namespace:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: WaypointInstance
metadata:
  name: main
  namespace: myns
spec:
  waypointFor: All
  replicasManagement:
    mode: Static
    static:
      replicas: 2
  resourcesManagement:
    mode: VPA
    vpa:
      mode: InPlaceOrRecreate
      cpu:
        min: 100m
        max: 1000m
      memory:
        min: 128Mi
        max: 2048Mi
```

The controller provisions the waypoint infrastructure (Deployment, Service, Gateway, VPA, and a PDB — when the effective replica count is `>= 2`). The controller **does not** attach workloads to the waypoint. You can do that with the `istio.io/use-waypoint` label.

To attach all workloads and services in the namespace to the waypoint, run the following command:

```
d8 k label namespace myns istio.io/use-waypoint=main
```

To attach a single service or workload, run the following command:

```
d8 k -n myns label service myservice istio.io/use-waypoint=main
```

Disabling ambient mesh

> ⚠ Before disabling ambient mode, delete all WaypointInstance resources. With ambient mode disabled, the waypoint controller is not running and cannot reconcile or clean up waypoint resources. This leaves orphaned waypoints, which are reported by Deckhouse Platform (DP) in the `D8IstioActiveWaypointsWithAmbientDisabled` alert.

To disable the ambient mode, follow these steps:

1. Check if any WaypointInstance resources remain and delete them as necessary using the following commands:

   ```
   d8 k get waypointinstance -A
   d8 k -n myns delete waypointinstance main
   ```

2. Disable the ambient mode by setting `ambient.enabled` to `false` in the module configuration.

Control the data-plane behavior

Jobs with an Istio sidecar

The module doesn’t inject the sidecar into Job Pods unless the Pod template asks for it explicitly:

```
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
```

This example assumes the namespace is labeled for Istio injection. In a namespace without an injection label, use the Pod template label `sidecar.istio.io/inject: "true"` instead of the annotation. For a CronJob, put these settings under `spec.jobTemplate.spec.template.metadata`.

A regular sidecar, which Istio 1.25 injects by default, keeps running after the application container exits and prevents the Job from completing. If the Pod uses a regular sidecar, for example with Istio 1.25, also add the `sidecar.istio.io/nativeSidecar: "true"` annotation to the Pod template.

Prevent istio-proxy from terminating before the main application’s connections are closed

With native sidecars, which Istio 1.27 and newer inject by default, `istio-proxy` stops only after the application containers have exited: it receives SIGTERM after them, and the application keeps network access while shutting down.

With a regular sidecar, which Istio 1.25 injects by default, all containers in a Pod, including `istio-proxy`, receive SIGTERM simultaneously. Applications that need network access while shutting down lose it when `istio-proxy` stops first. For these Pods, add a preStop hook to `istio-proxy`. The hook waits until the application has no network sockets left in the Pod’s network namespace:

```
annotations:
  inject.istio.io/templates: "sidecar,d8-hold-istio-proxy-termination-until-application-
stops"
```

The `d8-hold-istio-proxy-termination-until-application-stops` injection template works only with Istio 1.25. With Istio 1.27 and newer, it does nothing, even for Pods with a regular sidecar. Use a native sidecar for these Pods instead.

Telemetry API for mesh metrics and access logs

Istio Telemetry API (`telemetry.istio.io`) is the recommended way to configure the collection of data on the operation of services (metrics, access logs, tracing providers) in the mesh together with `meshConfig`.

The module can run in two modes, controlled by `telemetryAPI.enabled`:

Mode	Behaviour

Mode	Behaviour
`false` (default)	Classic path: full `telemetry.v2` in the Istio Operator / `Istio` resource (including Sail's `telemetry.v2.prometheus`). The module always deploys `Telemetry` `d8-main` in `d8-istio` for stdout access logs only; there is no `spec.metrics` / `spec.tracing` on that object and no `defaultProviders.metrics`
`true`	Telemetry API path: `meshConfig.defaultProviders.metrics` selects the built-in Prometheus provider; `telemetry.v2` integrations are toggled off; the same `Telemetry` `d8-main` gains `spec.metrics` (and optional `spec.tracing` via `deckhouse-tracing` when `tracing.collector` is configured). Access log format comes from `dataPlane.accessLog`

Enabling Telemetry API mode

Apply a `ModuleConfig` for the Istio module (or the same structure in the cluster config):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 1
  enabled: true
  settings:
    telemetryAPI:
      enabled: true
```

Wait until the `Istio` resource in `d8-istio` reconciles and workloads have picked up the new config (restart application pods if dashboards stay empty after traffic was sent).

Verifying metrics and logs

Generate traffic between meshed pods, then on a pod with `istio-proxy`:

```
# Prometheus text from the sidecar admin API (istio-proxy has pilot-agent, not curl).
istio_proxy_pod="$(
  d8 k -n my-namespace get pods -l app=my-app -o jsonpath='{.items[0].metadata.name}'
)"
d8 k exec -n my-namespace "${istio_proxy_pod}" -c istio-proxy -- \
  /usr/local/bin/pilot-agent request GET stats/prometheus | head
```

Example of successful output:

```
# TYPE istio_requests_total counter
istio_requests_total{...} 12
istio_request_duration_milliseconds_bucket{...} 12
istio_request_bytes_bucket{...} 12
istio_response_bytes_bucket{...} 12
```

If you see series such as `istio_requests_total`, metrics are wired correctly.

Prometheus scraping and Grafana

The module creates a `PodMonitor` for sidecar metrics when the `operator-prometheus` module is enabled. Monitored namespaces are derived automatically from mesh membership (istio-injected workloads); you can exclude a namespace from scraping with label `istio.deckhouse.io/discard-metrics: "true"` on the Namespace.

If workload dashboards stay empty while control plane dashboards work, confirm that:

- workloads run with the Istio sidecar and label `service.istio.io/canonical-name` is present on pods;
- the namespace is not marked with `istio.deckhouse.io/discard-metrics: "true"`.

Extra `Telemetry` policies (optional)

Prefer additional `Telemetry` objects that use a workload selector (or `targetRef`) to pin policies to matching pods. Two or more selector-less `Telemetry` resources in the same namespace are invalid—Istio reports IST0160 because the effective policy becomes ambiguous rather than cleanly “merged”. The module already ships `d8-main` in `d8-istio` as the single mesh-wide defaults object; avoid placing another selector-less `Telemetry` there unless you are intentionally replacing chart output.

Beyond that, scoped policies—for example metrics per namespace—look like:

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: team-a-prometheus-defaults
  namespace: team-a
spec:
  metrics:
  - providers:
    - name: prometheus
```

For tag removal, disabling specific metrics or modes, follow Customizing Istio metrics with Telemetry API.

Tracing with Telemetry API

`tracing.collector` is the single ModuleConfig entry for mesh-wide trace export (Zipkin or OpenTelemetry).

- `telemetryAPI.enabled: false` plus `tracing.enabled` `true`: Legacy only: `meshConfig.defaultConfig.tracing.zipkin` from `tracing.collector.zipkin.address` (`host:port`, Jaeger Zipkin port `9411`). OTLP (OpenTelemetry Protocol) export requires Telemetry API mode.
- `telemetryAPI.enabled: true` plus `tracing.enabled: true`: The module registers `deckhouse-tracing` when `tracing.collector.opentelemetry` has `service` and `port`, or when `tracing.collector.zipkin.address` is set (OpenTelemetry wins if both are configured). Legacy `defaultConfig.tracing` is omitted; mesh-wide `Telemetry` `d8-main` gets `spec.tracing` (`tracing.sampling` → `randomSamplingPercentage`, default `1.0`).

To send traces from some workloads to another OpenTelemetry or Zipkin backend, register it in `dataPlane.extensionProviders` and reference it from a Telemetry with `selector` in the application namespace. Do not create a second Telemetry without a selector in `d8-istio` (IST0160). A proxy sends spans to a single tracing provider, so these workloads send traces to that backend instead of `deckhouse-tracing`. Other exporters, such as SkyWalking, cannot be registered. See Distributed tracing with Telemetry API.

Example — Telemetry API bundle + bundled Zipkin/Jaeger collector

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 1
  enabled: true
  settings:
    telemetryAPI:
      enabled: true
    tracing:
      enabled: true
      sampling: 25
      collector:
        zipkin:
          address: "jaeger-collector.observability.svc.cluster.local:9411"
```

Roll out the `Istio` manifests in `d8-istio`; confirm workloads pick up telemetry before blaming dashboards.

Kiali

To see traces inside Kiali UI, configure `tracing.kiali` (Jaeger URL + cluster-internal gRPC endpoint) whenever Kiali is enabled.

Example — mesh-wide OTLP via ModuleConfig

Deploy a Collector reachable from the mesh, then enable Telemetry API mode and point `tracing.collector.opentelemetry` at it. The module adds extension provider `deckhouse-tracing` and `spec.tracing` on `d8-main`—do not patch the generated `Istio` `meshConfig` for mesh-wide OTLP.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 1
  enabled: true
  settings:
    telemetryAPI:
      enabled: true
    tracing:
      enabled: true
      sampling: 10
      collector:
        opentelemetry:
          service: opentelemetry-collector.observability.svc.cluster.local
          port: 4317
```

For HTTP OTLP, add `collector.opentelemetry.http.path` (and optional `timeout`) per `tracing.collector.opentelemetry.http`.

For per-workload overrides, create a Telemetry with `selector` in the application namespace and reference `deckhouse-tracing` or a provider registered in `dataPlane.extensionProviders`.

Example — tracing only for selected workloads

Use `selector` (or `targetRef`) so Telemetry targets only matching pods—the pattern below is IST0160-safe. Within a single namespace, `Telemetry` without selectors can still exist at most once; do not pile multiple selector-less manifests there.

Register the provider in `dataPlane.extensionProviders`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 1
  enabled: true
  settings:
    dataPlane:
      extensionProviders:
      - name: jaeger-zipkin
        zipkin:
          service: jaeger-collector.observability.svc.cluster.local
          port: 9411
```

Reference it from a Telemetry with `selector`:

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: checkout-tracing
  namespace: shop
spec:
  selector:
    matchLabels:
      app: checkout
  tracing:
  - providers:
    - name: jaeger-zipkin
    randomSamplingPercentage: 100.0
```

Example — disable span export for ingress-only namespaces

In DP, when the `ingress-nginx` module is enabled, the Istio chart creates `Telemetry` `ingress-nginx-disable-span-reporting` in `d8-ingress-nginx` with `tracing.disableSpanReporting` so Ingress controller pods with `istio-proxy` stop exporting spans. For other namespaces:

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: no-tracing-example
  namespace: my-namespace
spec:
  tracing:
  - disableSpanReporting: true
```

Rolling back to the legacy telemetry stack

```
spec:
  settings:
    telemetryAPI:
      enabled: false
```

The module-managed `Telemetry` objects for this mode disappear on the next sync; Istio restores the full `telemetry.v2` configuration.

Debugging Istio with istioctl from the debug container

The DP debug container includes versioned `istioctl` binaries. Use it when you need to inspect Istio configuration, run analyzers, or retrieve Envoy proxy configuration from application Pods.

Before starting the debug container, create a dedicated ServiceAccount and grant it the permissions required by the `istioctl` commands you want to run. For example, the following manifest grants permissions that allow running the `istioctl proxy-config` commands for Pods in a single application namespace:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: istioctl-debug
  namespace: <debug-namespace>
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: istioctl-debug
  namespace: <target-namespace>
rules:
  - apiGroups: [""]
    resources:
      - pods
    verbs:
      - get
      - list
  - apiGroups: [""]
    resources:
      - pods/portforward
    verbs:
      - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: istioctl-debug
  namespace: <target-namespace>
subjects:
  - kind: ServiceAccount
    name: istioctl-debug
    namespace: <debug-namespace>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: istioctl-debug
```

Replace `<debug-namespace>` with the namespace where the temporary debug Pod will be created, and `<target-namespace>` with the namespace of the application Pod you want to inspect. Create the Role and RoleBinding resources for every target namespace where `istioctl` must access Pods.

This RBAC manifest is intended for commands that address a Pod directly, for example, to a resource like `<pod-name>.<target-namespace>`. If you use typed resource names such as `deployment/<name>`, grant additional read access to those resource types so `istioctl` can resolve them to Pods.

> ⚠ Creating Pods in system namespaces such as `d8-system` and using system ServiceAccounts such as `deckhouse` usually requires the `cluster-admin` level of privileges. Use a dedicated ServiceAccount with the minimum required permissions instead.

Start a temporary debug Pod with the built-in debug image:

```
IMG="$(d8 k -n d8-system get cm debug-container -o jsonpath='{.data.image}')"

d8 k -n <debug-namespace> run istioctl-debug \
  --rm -it \
  --restart=Never \
  --image="$IMG" \
  --overrides='{"spec":{"serviceAccountName":"istioctl-
debug","automountServiceAccountToken":true}}' \
  -- bash
```

Select the minor version of Istio used by the target control plane:

```
export ISTIOCTL_VERSION=1.29
```

Available values are `1.25`, `1.27`, and `1.29`. You can also run a specific binary directly: `istioctl-1.25`, `istioctl-1.27`, or `istioctl-1.29`.

Example:

```
istioctl pc all <pod-name>.<target-namespace>
```

The `istioctl pc` commands require a target Pod with an injected `istio-proxy` sidecar. If the target Pod has no sidecar, Envoy admin port `15000` will not be available.

⚠ The RBAC manifest above is not enough to run `istioctl analyze` or `istioctl analyze -A`. These commands require additional read-only access to namespaces and to the Kubernetes and Istio resources covered by the analyzers. Grant such access separately according to your security policy.

CNIPlugin application traffic redirection mode restrictions

The restriction described below applies only to regular sidecars, which Istio 1.25 injects by default. A native sidecar starts before the application init containers, so their traffic goes through the mesh like the rest of the Pod's traffic.

Unlike the `InitContainer` mode, the redirection setting is done at the moment of Pod creating, not at the moment of triggering the `istio-init` init container. This means that application init containers will not be able to interact with other services because all traffic will be redirected to the `istio-proxy` sidecar container, which is not yet running. Workarounds:

- Run the application init container from the user with uid `1337`. Requests from this user are not intercepted under Istio control.
- Exclude a service IP address or port from Istio control using the `traffic.sidecar.istio.io/excludeOutboundIPRanges` or `traffic.sidecar.istio.io/excludeOutboundPorts` annotations.

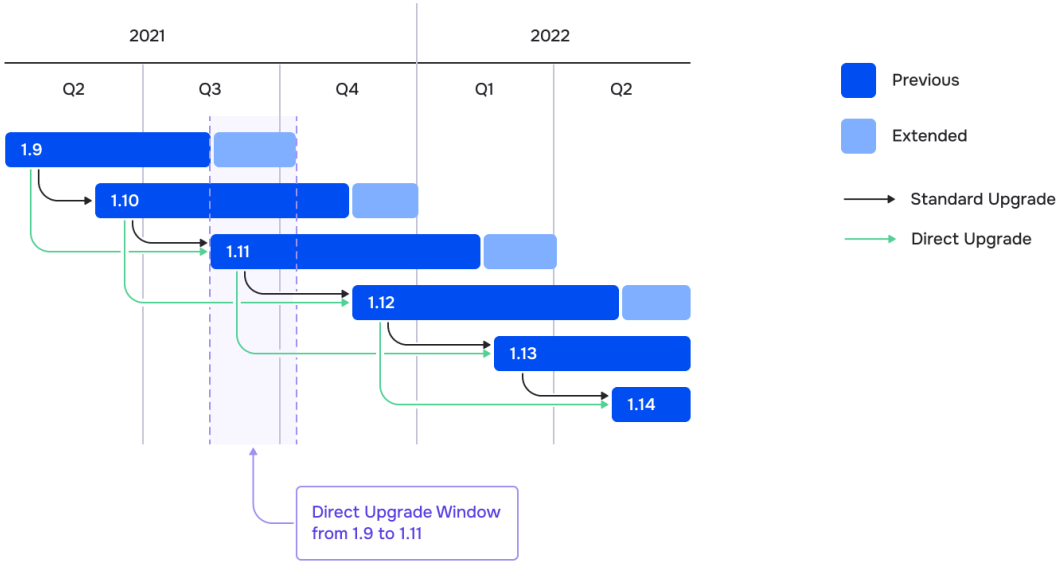
⚠ Each of the workarounds removes traffic from Istio's control and disables encryption between application services.

UID `1337` is reserved by Istio for the `istio-proxy` sidecar container. Do not run application containers with this UID — their traffic bypasses Istio entirely (no routing rules, mTLS, or telemetry). Use UID `1337` only in init containers when network requests are required before the sidecar is ready.

Upgrading Istio

Upgrading Istio control-plane

- DP allows you to install different control-plane versions simultaneously:
  - A single global version to handle namespaces or Pods with indifferent version (namespace label `istio-injection: enabled`). It is configured by the globalVersion parameter.
  - Additional versions handle namespaces or Pods with explicitly configured versions (`istio.io/rev: v1x25` label for namespace or Pod). They are configured by the `additionalVersions` parameter. The `istio.io/rev` label on a Pod is taken into account only if its namespace has neither the `istio-injection` nor the `istio.io/rev` label.
- Istio declares backward compatibility between data-plane and control-plane in the range of two minor versions:



- Upgrade algorithm (for example, from `1.25` to `1.27`):
  - Configure the target version in the additionalVersions parameter (`additionalVersions: ["1.27"]`).
- Wait for the corresponding pod `istiod-v1x27-xxx-yyy` to appear in the `d8-istio` namespace.
- For every application namespace with Istio enabled:
  - Change the `istio-injection: enabled` label to `istio.io/rev: v1x27`.
  - Recreate the Pods in the namespace one at a time while monitoring application health.
- Reconfigure `globalVersion` to `1.27` and remove the `additionalVersions` configuration.
- Make sure, the old `istiod` pod has gone.
- Change application namespace labels to `istio-injection: enabled`.

To find all Pods with the old Istio revision (version 1.25 in this example), execute the command:

```
d8 k get pods -A -o json | jq --arg revision "v1x25" \
  '.items[] | select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson
|
    .revision == $revision) | .metadata.namespace + "/" + .metadata.name'
```

Auto upgrading istio data-plane

To automate istio-sidecar upgrading, set a label `istio.deckhouse.io/auto-upgrade="true"` on the application `Namespace` or on the individual resources — `Deployment`, `DaemonSet` or `StatefulSet`.

Automatic upgrading is triggered when the current data-plane version of a Pod with an istio-sidecar differs from the desired version. Adding a version to the `additionalVersions` parameter does not restart application Pods by itself. A mismatch usually appears in the following cases:

- The `globalVersion` parameter changed for a namespace that uses the global Istio version (`istio-injection=enabled` or `istio.io/rev=default`).
- The `istio.io/rev` label changed on a Namespace, or on a Pod in a namespace without the `istio-injection` and `istio.io/rev` labels.
- The patch version of the installed control plane was updated.

Before restarting a workload, the module checks that the corresponding control plane is installed and ready. Then the module adds or updates the `istio.deckhouse.io/full-version` annotation in `spec.template.metadata.annotations`, and Kubernetes performs a regular rollout. Within the same namespace, the module does not start upgrading the next workload until the previously upgraded workload is ready.

The `istio.deckhouse.io/auto-upgrade="true"` label must be set on the same entity that defines Istio usage for the workload:

- If injection is enabled at the namespace level with `istio-injection=enabled`, `istio.io/rev=<REVISION>`, or `istio.io/rev=default`, you can set `istio.deckhouse.io/auto-upgrade="true"` on the same Namespace.
- If sidecar injection is enabled at the workload or pod template level, for example with `sidecar.istio.io/inject="true"`, set `istio.deckhouse.io/auto-upgrade="true"` on the corresponding Deployment, DaemonSet, or StatefulSet.
- A namespace with only the `istio.deckhouse.io/auto-upgrade="true"` label does not enable automatic upgrading for a workload if injection is configured only at the workload or pod template level.

Automatic upgrading is supported only for Deployment, DaemonSet, and StatefulSet resources. Job, CronJob, standalone Pod, and custom controllers, including Kruise `AdvancedDaemonSet`, are not handled. If an ingress controller is managed by an `AdvancedDaemonSet`, the `istio.deckhouse.io/auto-upgrade="true"` label on that resource is ignored. Upgrade such ingress controllers manually following the “Istio control plane upgrade” procedure and the “Ingress NGINX integration” section.

Use the `D8IstioDataPlaneVersionMismatch` alert to detect Pods whose patch data-plane version differs from the control plane version — the scenario that automatic upgrading addresses. The `D8IstioActualDataPlaneVersionNotEqualDesired` alert indicates a revision mismatch and usually requires changing namespace or Pod labels before sidecars can be updated.

⚠ Do not delete the old Istio control plane manually during an upgrade. The old `istiod` and related resources are removed automatically after there are no sidecars connected to the old revision left in the cluster. Manual deletion can break the regular upgrade automation.

Customizing istio-proxy sidecar resource management

You can override the global istio-proxy sidecar resource limits for specific workloads by adding annotations to your application Pods.

Supported annotations

Use these Pod annotations to customize sidecar resources:

Annotation	Description	Example Value
`sidecar.istio.io/proxyCPU`	CPU request for sidecar	`200m`
`sidecar.istio.io/proxyCPULimit`	CPU limit for sidecar	`"1"`
`sidecar.istio.io/proxyMemory`	Memory request for sidecar	`128Mi`
`sidecar.istio.io/proxyMemoryLimit`	Memory limit for sidecar	`512Mi`

Configuration Examples

For Deployments:

```
apiVersion: apps/v1
kind: Deployment
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
         sidecar.istio.io/proxyCPU: 200m
         sidecar.istio.io/proxyCPULimit: "1"
         sidecar.istio.io/proxyMemory: 128Mi
         sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your deployment spec
```

For ReplicaSets:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
          sidecar.istio.io/proxyCPU: 200m
          sidecar.istio.io/proxyCPULimit: "1"
          sidecar.istio.io/proxyMemory: 128Mi
          sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your deployment spec
```

For Pod:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    sidecar.istio.io/proxyCPU: 200m
    sidecar.istio.io/proxyCPULimit: "1"
    sidecar.istio.io/proxyMemory: 128Mi
    sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your pod spec
```

⚠ All four parameters must be defined together - if you set any of these annotations, you must specify all four (`sidecar.istio.io/proxyCPU`, `sidecar.istio.io/proxyCPULimit`, `sidecar.istio.io/proxyMemory`, and `sidecar.istio.io/proxyMemoryLimit`).

Providing a custom CA via a Secret reference

Instead of pasting the CA certificate and key inline into the module configuration (via `ca.cert` and `ca.key`), you can reference an existing Secret with the `ca.secretRef` parameter. This lets you issue the CA with any preferred tooling, for example, cert-manager (self-signed issuer, Vault issuer, etc.).

The referenced Secret may be in one of the following formats (auto-detected):

- a cert-manager `kubernetes.io/tls` Secret with the `tls.crt`, `tls.key` and `ca.crt` keys;
- a native Istio `cacerts` Secret with the `ca-cert.pem`, `ca-key.pem`, `cert-chain.pem` and `root-cert.pem` keys.

The signing certificate must be a CA certificate (`isCA: true` / `basicConstraints = CA:TRUE`) whose key matches it.

Example: self-signed CA issued by cert-manager

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ca
  namespace: my-pki
spec:
  secretName: istio-ca
  commonName: istiod-ca
  isCA: true
  usages:
    - digital signature
    - key encipherment
    - cert sign
  issuerRef:
    kind: Issuer
    name: selfsigned-issuer
```

Example: intermediate CA issued by a Vault issuer

If you already run a HashiCorp Vault PKI, you can have cert-manager issue the Istio CA as an intermediate signed by your Vault root/intermediate. The resulting Secret has exactly the same layout and is auto-detected the same way — the only requirement is `isCA: true`.

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: my-pki
spec:
  vault:
    # Path of the intermediate PKI role that is allowed to issue CA certificates.
    path: pki_int/sign/istio-ca
    server: https://vault.example.com:8200
    # Vault must return the CA chain so that ca.crt contains the root.
    caBundle: <base64-encoded Vault CA bundle>
    auth:
      kubernetes:
        role: istio-ca-issuer
        mountPath: /v1/auth/kubernetes
        serviceAccountRef:
          name: cert-manager
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ca
  namespace: my-pki
spec:
  secretName: istio-ca
  commonName: istiod-ca
  # Mandatory: the issued certificate must itself be a CA so istiod can sign workload
certs.
  isCA: true
  duration: 87600h   # 10y, adjust to your rotation policy
  usages:
    - digital signature
    - key encipherment
    - cert sign
    - crl sign
  issuerRef:
    kind: Issuer
    name: vault-issuer
```

⚠ The Vault PKI role must permit issuing CA certificates (`isCA`), and the issuer must return the CA chain so that `ca.crt` contains the Vault root. Otherwise the module cannot determine the root and the leaf validation will fail, hard-blocking the rollout.

Then reference the resulting Secret in the module configuration (identical for both examples above):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 3
  enabled: true
  settings:
    ca:
      secretRef:
        name: istio-ca
        namespace: my-pki
```

i If `namespace` is omitted, the Secret is looked up in the `d8-istio` namespace.

⚠ If the referenced Secret is missing or malformed on the first resolution of this `secretRef`, the module will not proceed until the issue is fixed. It will not silently fall back to a self-signed CA — or to any CA that was in use before this `secretRef` was configured — to avoid unexpectedly running on an unrelated CA.

If the same `secretRef` was already resolved successfully and later becomes unavailable or invalid (for example, it is briefly deleted during rotation, or its chain expires), the module keeps using that last successfully-resolved CA instead of hard-blocking. It never switches to a different CA on its own — it only reuses the exact material istiod is already running with — so an already-working mesh is not regressed while the source is fixed. Repointing `secretRef` to a different Secret counts as a first resolution: if that new Secret cannot be resolved, the module hard-blocks rather than keeping the previous CA.

i The referenced Secret is re-resolved on a 5-minute schedule, so changes to the source Secret may propagate with a delay.

When using an intermediate CA, keep the root stable across rotations. The module publishes `root-cert.pem` (the `ca.crt` trust anchor) as the webhook `caBundle` and as the workload trust root, while the signing (intermediate) certificate may change. Rotating the intermediate under a stable root is seamless; changing the root is a mesh-wide trust change and requires the old and new roots to overlap in the trust bundle during the transition.

Reverting to a self-signed CA

Removing `ca.secretRef` (or inline `ca.cert`) from the module configuration does not switch the mesh back to an auto-generated self-signed CA on its own. The module intentionally keeps the last-published CA — stored in the `d8-istio/cacerts` Secret — so an already-working mesh is never rotated implicitly: rotating the mesh root breaks mTLS cluster-wide until every workload re-trusts the new root.

To deliberately return to a self-signed CA:

1. Remove the `ca.secretRef` (or `ca.cert`) configuration from the `istio` ModuleConfig.
2. Delete the module-owned `d8-istio/cacerts` Secret:

   ```
   d8 k -n d8-istio delete secret cacerts
   ```

3. Restart the Deckhouse pod so the module drops the last-published CA it holds in memory:

   ```
   d8 k -n d8-system rollout restart deployment deckhouse
   ```

⚠ This is a mesh-wide trust change and triggers a rollout: istiod republishes the new root as the webhook `caBundle` and workloads must re-trust it. Perform it during a maintenance window.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The keepalived module

> The module lifecycle stage: General Availability

> ⚠ This module does not work with the cilium module.

This module (managed via custom resources) is intended for configuring keepalived clusters on nodes.

Third-party components

List of third-party software used in the `keepalived` module:

- keepalived 2.2.7

  License: GNU General Public License v2.0

  Provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

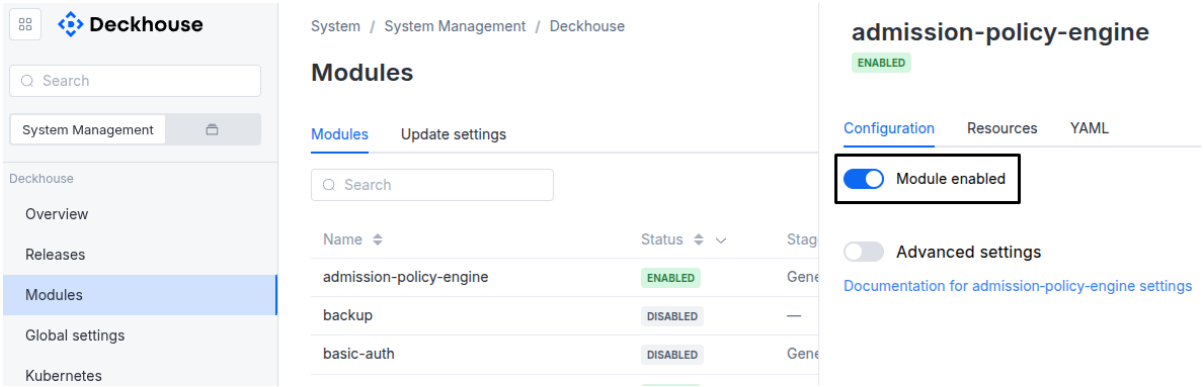
> The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `keepalived` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable keepalived
```

- **Using ModuleConfig `keepalived`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `keepalived` (create it if necessary);

  Example of a manifest to enable module `keepalived`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: keepalived
spec:
  enabled: true
```

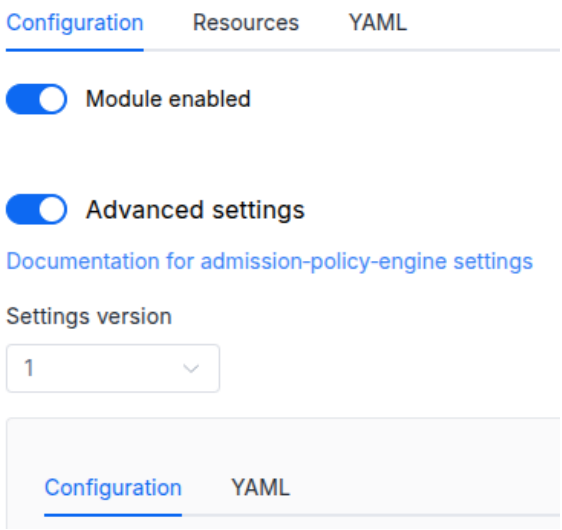
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `keepalived` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `keepalived` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `keepalived`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `keepalived` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc keepalived
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `keepalived` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `keepalived`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: keepalived
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Keepalived clusters are configured using the custom resource.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

KeepalivedInstance

Scope: Cluster
Version: v1alpha1

The custom resource for configuring keepalived clusters.

Each `KeepalivedInstance` custom resource defines a single keepalived cluster.

`metadata.name` — the name of the cluster (used in pod names).

- **spec**
  object

  Required value

  - **spec.nodeSelector**
    object

    Required value

    Defines the `nodeSelector` parameter for keepalived pods.

    Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

  - **spec.tolerations**
    array of objects

    Defines `tolerations` for keepalived pods.

    Format: the standard toleration list. Instance pods inherit this field as is.

    - **spec.tolerations.effect**
      string

      Defines the taint’s effect this toleration is associated with (it matches any effect if empty).

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **spec.tolerations.key**
      string

      Defines the taint’s key this toleration is associated with (it matches any key if empty).

If the key is omitted (emplty), then `operator` must be set to `Exists` (which means it matches any `value` and `key`).

- **spec.tolerations.operator**
  string

  Defines how the key and its value are linked — whether the key must be `Equal` to the value or can have any value.

  Setting it to `Exists` is equivalent to the value having any value so that the pod with the specified toleration can match the corresponding taint.

  Default: `Equal`

  Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer

  Specifies the period (in seconds) that the scheduler waits before evicting a Pod from a node if the toleration no longer matches the taint (**only** applicable to the `NoExecute` effect, otherwise ignored).

  If the parameter **is not set**, the Pod will not be evicted from the node if the toleration no longer matches the taint. If the parameter **is set to zero** (or negative), the Pod will be evicted immediately if the toleration no longer matches the taint.

  The default is: not set.

- **spec.tolerations.value**
  string

  The value that the toleration must match.

  Must be empty if the operator parameter is set to `Exists`.

- **spec.vrrpInstances**
  array of objects

  Required value

  A list of VRRP instances in the keepalived cluster. Essentially, it is a list of address groups that migrate between servers simultaneously and cannot work separately.

  Do not confuse `vrrpInstance` with `KeepalivedInstance`. `vrrpInstance` is a component of `KeepalivedInstance`.

  The `keepalived` module configures VRRP instances so that all addresses (all groups) are evenly distributed across all servers (instead of gathering together on the same node).

  - **spec.vrrpInstances.id**
    integer

    Required value

    A unique **cluster-wide** identifier of a VRRP group.

    Do not use the same ID in different `KeepalivedInstance` instances unless you have a perfect reason to do so.

    Allowed values: `1 <= X <= 255`

  - **spec.vrrpInstances.interface**
    object

    Required value

    Defines the strategy of detecting interface for the service VRRP traffic on the node.

    - **spec.vrrpInstances.interface.detectionStrategy**
      string

      You can select one of the following three interface detecting strategis:

      - `Name` — set the interface name explicitly using the `spec.vrrpInstances[].interface.name` parameter. In this case, all nodes must have the same interface that serves the correct network (e.g., eth0).
      - `NetworkAddress` — find an interface with an IP from this subnet on the node and use it.
      - `DefaultRoute` — use the interface for the default route on the node (254 “main” in the table).
    - **spec.vrrpInstances.interface.name**

string

The name of the interface for the service VRRP traffic if the `Name` `detectionStrategy` is used.

A mandatory parameter if `detectionStrategy` = `Name`.

- **spec.vrrpInstances.interface.networkAddress**
  string

  The address of the subnet to use for determining an interface if `detectionStrategy` = `NetworkAddress`.

  A mandatory parameter if `detectionStrategy` = `NetworkAddress`.

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

  Example:

  ```
  networkAddress: 192.168.42.0/24
  ```

- **spec.vrrpInstances.preempt**
  boolean

  Defines if the IP is passed to a node after its recovery from a failure.

  It makes sense not to pass the IP if you have just one `vrrpInstance` to preserve connections. If you have a lot of groups and a lot of traffic, it is better to pass it to prevent the concentration of all groups on one node.

  The default is — `true`, that is, the IP will passed to a node after its recovery.

- **spec.vrrpInstances.virtualIPAddresses**
  array of objects

  Required value

  A list of IP addresses that will “jump” between the servers **in tandem**

  - **spec.vrrpInstances.virtualIPAddresses.address**
    string

    Required value

    One of the group addresses.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

    Example:

    ```
    address: 192.168.42.15/32
    ```

  - **spec.vrrpInstances.virtualIPAddresses.interface**
    object

    Not required value.

    An interface for binding a virtual IP address on a node (similar to `spec.vrrpInstances[].interface`).

    If omitted, the main service interface defined in `spec.vrrpInstances[].interface` will be used.

    - **spec.vrrpInstances.virtualIPAddresses.interface.detectionStrategy**
      string

      You can select one of the following three interface detecting strategis ti bind an IP address:

      - `Name` — set the interface name explicitly using the `spec.vrrpInstances[].virtualIPAddresses[].interface.name` parameter. In this case, all nodes must have the same interface that serves the correct network (e.g., eth0).
      - `NetworkAddress` — find an interface with an IP from this subnet on the node and use it.
      - `DefaultRoute` — use the interface for the default route on the node (254 “main” in the table).
    - **spec.vrrpInstances.virtualIPAddresses.interface.name**
      string

A name of the interface to bind a virtual IP address if `virtualIPaddresses[].detectionStrategy = Name` is used.

A mandatory parameter if `virtualIPaddresses[].detectionStrategy = Name`.

- **spec.vrrpInstances.virtualIPAddresses.interface.networkAddress** string

  The address of the subnet to use for determining an interface if `virtualIPaddresses[].detectionStrategy = NetworkAddress`.

  A mandatory parameter if `virtualIPaddresses[].detectionStrategy = NetworkAddress`.

  Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

  Example:

```
networkAddress: 192.168.42.0/24
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

keepalived: FAQ

> The module lifecycle stage: General Availability

How to manually switch keepalived?

1. Enter the desired pod using a debug container with a shared process namespace: `d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived`.
2. Edit the configuration file `vim /proc/1/root/etc/keepalived/keepalived.conf`, replace the value in the `priority` line with <number of keepalived pods + 1> or set a value higher than the current VRRP master (e.g., `255`).
3. Apply settings – send a signal to reload the configuration: `kill -HUP 1`.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Three public IP addresses

Suppose there are three public IP addresses on three front nodes. Each virtual IP address is placed in a separate VRRP group. Thus, each address “jumps” independently of the others, and if there are three nodes in the cluster with the `node-role.deckhouse.io/frontend: ""` labels, then each IP gets its own MASTER node.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Mandatory.
    node-role.deckhouse.io/frontend: ""
  tolerations:  # Optional.
  - key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
  vrrpInstances:
  - id: 1 # The unique cluster ID.
    interface:
      detectionStrategy: DefaultRoute # The card with the default route is used as a
service network one.
    virtualIPAddresses:
    - address: 42.43.44.101/32
      # The interface parameter is omitted since IP addresses are based on the cards that
service VRRP traffic.
  - id: 2
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.102/32
  - id: 3
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.103/32
```

Suppose there is a gateway with a pair of IP addresses for LAN and WAN. In the case of the gateway, the private and public IPs are bind together, and they will “jump” between the nodes in tandem. In the below example, the VRRP traffic is routed through the LAN interface. It can be detected using the NetworkAddress method (assuming that each node has an IP belonging to this subnet).

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
  - key: node-role.deckhouse.io/mygateway
    operator: Exists
  vrrpInstances:
  - id: 4 # Since "1", "2", "3" IDs are used in the "front" KeepalivedInstance above.
    interface:
      detectionStrategy: NetworkAddress
      networkAddress: 192.168.42.0/24
    virtualIPAddresses:
    - address: 192.168.42.1/24
      # In this case, we have already detected the local network (above); thus, the
interface parameter can be safely omitted.
    - address: 42.43.44.1/28
      interface:
        detectionStrategy: Name
        name: ens7 # we use the fact that an interface for public IPs is called "ens7" on
all nodes
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The kube-dns module

> The module lifecycle stage: General Availability

The module installs CoreDNS components for managing DNS in the Kubernetes cluster.

> **Caution.** The module deletes all the previously installed kubeadm Deployments, ConfigMaps as well as RBAC for CoreDNS. When deploying your own CoreDNS, avoid using the names `coredns` or `system:coredns` for any resources (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Use alternative names like `infra-dns` to prevent automatic removal by Deckhouse Platform.

Third-party components

List of third-party software used in the `kube-dns` module:

- CoreDNS

  License: Apache License 2.0

  A DNS server/forwarder that chains plugins.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

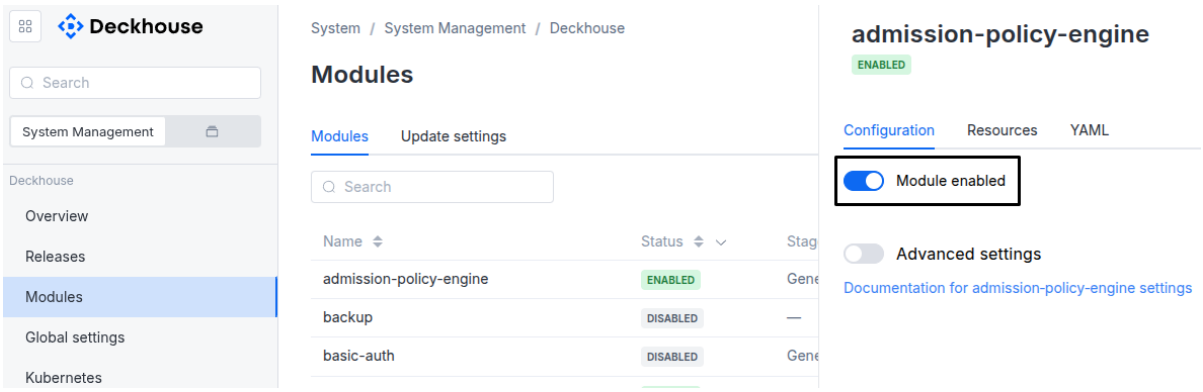
The module has 1 alert.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `kube-dns` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable kube-dns
  ```

- **Using ModuleConfig `kube-dns`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `kube-dns` (create it if necessary);

  Example of a manifest to enable module `kube-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: kube-dns
  spec:
    enabled: true
  ```

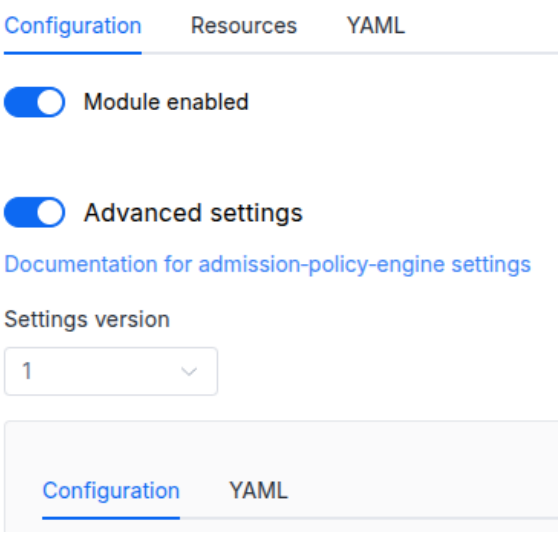
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `kube-dns` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `kube-dns` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `kube-dns`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `kube-dns` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc kube-dns
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `kube-dns` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `kube-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: kube-dns
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.clusterDomainAliases**
    array of strings

    A list of cluster domain aliases; these are resolved on par with `global.discovery.clusterDomain`.

    Please note: the domain alias must not match the domain used in the DNS name template in the publicDomainTemplate parameter.

    - **Element of the array**
      string

      Pattern: `^[0-9a-zA-Z\.-]+$`
  - **settings.enableLogs**
    boolean

    Enable CoreDNS logging.

    Default: `false`
  - **settings.hosts**
    array of objects

    Not required value.

    A static list of hosts similar to that of `/etc/hosts`.

    - **settings.hosts.domain**
      string

      Pattern: `^[0-9a-zA-Z\.-]+$`
    - **settings.hosts.ip**
      string

      Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}$`
  - **settings.stubZones**
    array of objects

A list of additional zones CoreDNS should be authoritative for.

Default: `[]`

Example:

```
stubZones:
- zone: example.com
  upstreamNameservers:
  - 8.8.8.8
  cacheTTLSeconds: 3600
  transportProtocolMode: PreferUDP
```

- **settings.stubZones.cacheTTLSeconds**
  integer

  Max TTL in seconds for NOERROR responses.

  Default: `30`

  Allowed values: `1 <= X <= 3600`

- **settings.stubZones.transportProtocolMode**
  string

  The mode of the transport protocol for communicating with the upstream DNS server:

  - `PreferUDP` — UDP is used, even if the original request came over TCP. Note that if the DNS response from the upstream DNS server exceeds 512 bytes and has been truncated and marked with the TC bit (RFC 1035), then it will be resent over TCP (RFC 5966). The request will not be resent over TCP if there are other UDP issues.
  - `ForceTCP` — Always uses TCP, even if the original request is over UDP.
  - `KeepOriginal` — Uses the same protocol (TCP or UDP) as the original request.

  Default: `PreferUDP`

  Allowed values: `PreferUDP`, `ForceTCP`, `KeepOriginal`

- **settings.stubZones.upstreamNameservers**
  array of strings

  Required value

  A list of IP addresses of recursive DNS servers that CoreDNS will use to resolve domains in this zone.

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?$`

- **settings.stubZones.zone**
  string

  Required value

  The CoreDNS zone.

  Pattern: `^[0-9a-zA-Z\.\-_]+$`

  Example:

  ```
  zone: consul.local
  ```

- **settings.transportProtocolMode**
  string

  The mode of the transport protocol for communicating with the upstream DNS server:

  - `PreferUDP` — UDP is used, even if the original request came over TCP. Note that if the DNS response from the upstream DNS server exceeds 512 bytes and has been truncated and marked with the TC bit (RFC 1035), then it will be resent over TCP (RFC 5966). The request will not be resent over TCP if there are other UDP issues.
  - `ForceTCP` — Always uses TCP, even if the original request is over UDP.
  - `KeepOriginal` — Uses the same protocol (TCP or UDP) as the original request.

  Default: `PreferUDP`

  Allowed values: `PreferUDP`, `ForceTCP`, `KeepOriginal`

- **settings.upstreamNameservers**
  array of strings

  A list of IP addresses of recursive DNS servers that CoreDNS will use to resolve external domains.

  By default, the `/etc/resolv.conf` list is used.

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?$`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

How to change cluster domain with minimal downtime?

> ⚠ **Before you start.** The cluster domain must already be managed from the `control-plane-manager` ModuleConfig. If the `D8ObsoleteClusterDomainInClusterConfiguration` alert is firing, it is still kept in `ClusterConfiguration`; finish the migration first:
>
> 1. Copy the current value into `spec.settings.network.clusterDomain` of the `control-plane-manager` ModuleConfig. Keep the value exactly as it is now: admission rejects a mismatch on the initial move.
>
>    ```
>    d8 k edit mc control-plane-manager
>    ```
>
> 2. Remove `clusterDomain` from `ClusterConfiguration`:
>
>    ```
>    d8 system edit cluster-configuration
>    ```
>
> 3. Make sure the alert has cleared.

Add a new domain and retain the previous one. To do this, modify the configuration parameters:

1. In the controlPlaneManager.apiserver:
   - controlPlaneManager.apiserver.certSANs,
   - apiserver.serviceAccount.additionalAPIAudiences,
   - apiserver.serviceAccount.additionalAPIIssuers.

   Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
  version: 3
  settings:
    apiserver:
      certSANs:
       - kubernetes.default.svc.<old clusterDomain>
       - kubernetes.default.svc.<new clusterDomain>
      serviceAccount:
        additionalAPIAudiences:
        - https://kubernetes.default.svc.<old clusterDomain>
        - https://kubernetes.default.svc.<new clusterDomain>
        additionalAPIIssuers:
        - https://kubernetes.default.svc.<old clusterDomain>
        - https://kubernetes.default.svc.<new clusterDomain>
```

2. In the kubeDns.clusterDomainAliases:

   Example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: kube-dns
   spec:
     version: 1
     enabled: true
     settings:
       clusterDomainAliases:
         - <old clusterDomain>
         - <new clusterDomain>
   ```

3. Wait until the `kube-apiserver` has restarted.

   ```
   d8 k -n kube-system get pods -l component=kube-apiserver
   ```

4. Set `spec.settings.network.clusterDomain` to the new domain:

   ```
   d8 k edit mc control-plane-manager
   ```

5. Restart deckhouse pods:

   ```
   kubectl -n d8-system rollout restart deployment deckhouse
   ```

⚠ **Important.** In kubernetes, controllers in the cluster use advanced ServiceAccount tokens to interact with API server. These tokens have extra fields `iss:` and `aud:` that contain `clusterDomain` (e.g. `"iss": "https://kubernetes.default.svc.cluster.local"`). After changing the clusterDomain, the API server will start issuing tokens with the new service-account-issuer, but thanks to the configuration of additionalAPIAudiences and additionalAPIIssuers, the apiserver will continue to accept the old tokens. After 48 minutes (80% of 3607 seconds), Kubernetes will begin to refresh the issued tokens, and the new service-account-issuer will be used for the updated tokens. After 90 minutes (3607 seconds plus a short buffer) following the kube-apiserver restart, you can remove the serviceAccount configuration from the control-plane-manager configuration.

⚠ **Important.** In case you need to remove old domain from `clusterDomainAliases` in kube-dns configuration, recreation of all pods in cluster is required, so new pods will have correct search domain in `/etc/resolv.conf` file. This will lead to service downtime until pods are restarted.

```
d8 k delete pods --all-namespaces --all
```

⚠ **Important!** If you use istio module, you have to restart all the application pods under istio control after changing `clusterDomain`.

How to scale-up kube-dns replicas?

Deckhouse distributes kube-dns pods based on the following principles. It searches for nodes with the labels `node-role.deckhouse.io/` and `node-role.kubernetes.io/`, then applies the following rules:

- If there are nodes with the `kube-dns` role in the cluster, the number of replicas is calculated as the sum of these nodes and the master nodes, but not more than masters count plus 2.
- If kube-dns nodes are absent, the system searches for nodes with the `system` role, and the number of replicas is then determined as the sum of system nodes and master nodes, but not more than masters count plus 2.
- If only master nodes are present in the cluster, the number of kube-dns replicas will be equal to the number of masters.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

i The module lifecycle stage: General Availability

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
    - 8.8.8.8
    - 8.8.4.4
    hosts:
    - domain: one.example.com
      ip: 192.168.0.1
    - domain: two.another.example.com
      ip: 10.10.0.128
    stubZones:
    - zone: consul.local
      upstreamNameservers:
      - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
    - foo.bar
    - baz.qux
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The kube-proxy module

> The module lifecycle stage: General Availability

The module deletes the entire `kubeadm` kube-proxy set (`DaemonSet`, `ConfigMap`, `RBAC`) and installs its own.

For security reasons, for NodePort services, connections are only allowed to the nodes' InternalIP by default. You can lift this restriction using the `node.deckhouse.io/nodeport-bind-internal-ip: "false"` annotation.

Here is an example of a NodeGroup annotation:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
...
```

> **Note!** Following the addition, deletion, or changing the annotation, you have to restart kube-proxy Pods manually.
>
> **Note!** The kube-proxy module is automatically disabled when the cni-cilium module is enabled.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The metallb module

> Available with limitations in: **Core, SE, SE+, Certified Core/CSE Lite (1.73)**
>
> Available without limitations in: **Ultimate/EE**

> Included with limitations in extensions: **Advanced Networking**

> The module lifecycle stage: General Availability

This module implements the `LoadBalancer` mechanism for services in bare metal clusters.

Supports the following operating modes:

- **Layer 2 Mode** – introduces an improved load-balancing mechanism for bare metal clusters (compared to the standard L2 mode in MetalLB), enabling the use of multiple "public" IP addresses for cluster services.

- **BGP Mode** – fully based on the MetalLB solution.

Layer 2 mode

In Layer 2 mode, one or more nodes take responsibility for providing the service within the “public” network. From the network’s perspective, it appears as if each of these nodes has multiple IP addresses assigned to its network interface. Technically, this is achieved by the module responding to ARP requests for IPv4 services and NDP requests for IPv6 services. The primary advantage of Layer 2 mode is its versatility: it works in any Ethernet network without requiring specialized hardware.

Advantages of the module over the classic MetalLB

MetalLB in L2 mode allows ordering Service with `LoadBalancer` type, the operation of which is based on the fact that balancing nodes simulate ARP-responses from the “public” IP in a peering network. This mode has a significant limitation — only one balancing node handles all the incoming traffic of this service at a time. Therefore:

- The node selected as the leader for the “public” IP becomes a “bottleneck”, with no possibility of horizontal scaling.
- If the balancer node fails, all current connections will be dropped while switching to a new balancing node that will be selected as the leader.

This module helps to overcome these limitations. It introduces a new resource, MetalLoadBalancerClass, which allows associating a group of nodes with an IP address pool using a `nodeSelector`. Afterward, a standard Service resource of type `LoadBalancer` can be created, specifying the name of the corresponding MetalLoadBalancerClass. Additionally, annotations can be used to define the required number of IP addresses for L2 advertisement.

Thus:

- The application will receive not a single, but several (according to the number of balancer nodes) “public” IPs. These IPs will need to be configured as A-records for the application’s public domain. For further horizontal scaling, additional balancer nodes will need to be added, the corresponding Service will be created automatically, you just need to add them to the list of A-records for the application domain.
- If one of the balancer nodes fails, only part of the connections will fail over to the healthy node.

BGP Mode

Metallb in BGP mode provides an efficient and scalable way to expose `LoadBalancer` type Services in Kubernetes clusters running on bare metal. By utilizing the standardized BGP protocol, metallb seamlessly integrates into existing network infrastructure and ensures high availability of Services.

How metallb Works in BGP Mode

In BGP mode, metallb establishes BGP sessions with routers (or Top-of-Rack switches) and announces the IP addresses of `LoadBalancer` type services to them. This is accomplished as follows:

Configuration.

- Metallb is configured with a pool of IP addresses that it can assign to Services.
- BGP session parameters are defined: the Autonomous System (AS) number of the Kubernetes cluster, the IP addresses of the routers (peers), the AS number of the peers, and optionally, authentication passwords.
- For each IP address pool, specific announcement parameters can be set, such as community strings.

Establishing BGP Sessions.

- On each node of the Kubernetes cluster where metallb is running, the speaker component establishes BGP sessions with the specified routers.
- Routing information is exchanged between metallb and the routers.

Assigning IP Addresses to Services.

- When a Service of type `LoadBalancer` is created, metallb selects a free IP address from the configured pool and assigns it to the Service.
- The controller component tracks changes to Services and manages IP address assignments.

Announcing IP Addresses.

- After an IP address is assigned, the speaker on the node elected as the leader for that Service begins announcing the IP address over the established BGP sessions.
- The routers receive this announcement and update their routing tables, directing traffic for that IP address to the node that announced it.

Traffic Distribution.

- Routers use Equal-Cost Multi-Path (ECMP) or other load balancing algorithms to distribute traffic among

nodes announcing the same Service IP address.
- Inside the Kubernetes cluster, traffic arriving at a node is forwarded to the Service's pods using the mechanisms of the employed CNI (iptables/IPVS, eBPF programs, etc.).

Advantages of Using BGP

- **Standardized Protocol:** BGP is a widely used and well-established routing protocol.
- **Flexibility and Scalability:** BGP allows you to integrate your Kubernetes cluster into your existing network infrastructure and scale your services.
- **Load Distribution:** Using ECMP allows for efficient load distribution across cluster nodes.
- **Fault Tolerance:** If a node announcing an IP address fails, routers automatically redirect traffic to other nodes announcing the same IP.

Disadvantages of Using BGP

- **Configuration Complexity:** Configuring BGP can be more complex than configuring ARP/NDP announcements (in Layer 2 mode).
- **Network Equipment Requirements:** Routers must support BGP and ECMP.

Third-party components

List of third-party software used in the `metallb` module:

- MetalLB 0.15.2

  License: Apache License 2.0

  A load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> Available with limitations in: **Core, SE, SE+, Certified Core/CSE Lite (1.73)**
>
> Available without limitations in: **Ultimate/EE**

> Included with limitations in extensions: **Advanced Networking**

> The module lifecycle stage: General Availability

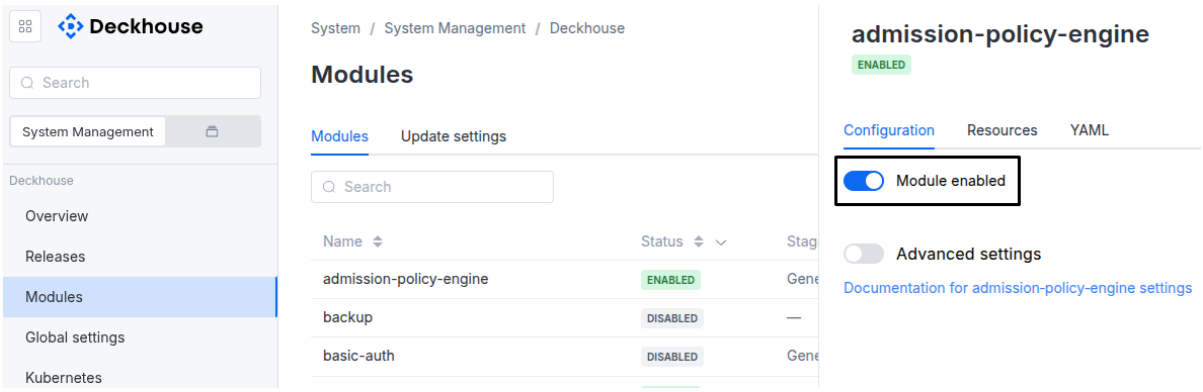
The module has 7 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `metallb` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable metallb
  ```

- **Using ModuleConfig `metallb`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `metallb` (create it if necessary);

  Example of a manifest to enable module `metallb`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: metallb
  spec:
    enabled: true
  ```

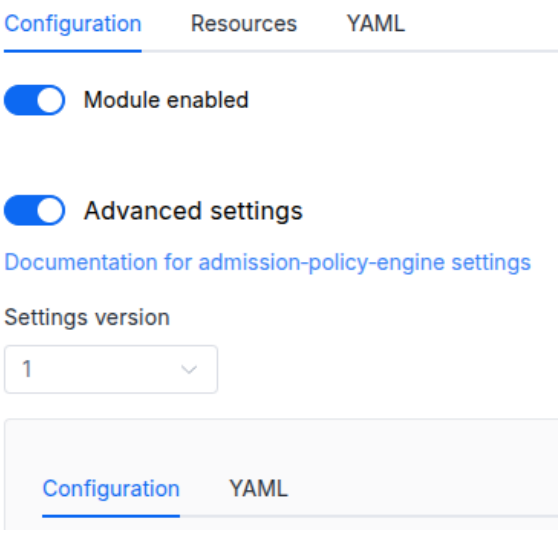
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `metallb` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `metallb` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `metallb`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `metallb` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc metallb
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `metallb` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `metallb`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: metallb
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Remove all `addressPool` elements with the `layer2` protocol.

- Updates from version **2** to **3**:

  Address pools and BGP settings have been moved to separate Custom Resources. Please remove the following fields from the `metallb` `ModuleConfig`:

  - `.settings.addressPools`
  - `.settings.bgpPeers`
  - `.settings.bgpCommunities`

Parameters

Schema version: 3

Example:

```
nodeSelector:
  node-role.deckhouse.io/control-plane: ''
```

- **settings**
  object
  - **settings.addressPools**
    Deprecated
    array of objects

    A list of IP ranges to assign to services.

    Format — a data array similar to that of MetalLB’s.

    Default: `[]`

    - **settings.addressPools.addresses**
      array of strings

Required value

A list of ranges, where each range can look like a subnet/mask or a numeric address range (with “-“ as a delimiter).

- **settings.addressPools.auto-assign**
boolean

  Auto-assign flag used to prevent metallb from automatic allocation for a pool.

  Default: `true`

- **settings.addressPools.avoid-buggy-ips**
boolean

  Prevents addresses ending with `.0` and `.255` to be used by a pool.

  Default: `false`

- **settings.addressPools.bgp-advertisements**
array of objects

  Defines BGP advertisements.

  - **settings.addressPools.bgp-advertisements.aggregation-length**
integer

    The aggregation-length advertisement option lets you “roll up” prefix into a larger one.

    Works for IPv4 addresses.

    Default: `32`

    Allowed values: `1 <= X`

  - **settings.addressPools.bgp-advertisements.communities**
array of strings

    Keys from the bgpCommunities parameter to be associated with the announcement.

    Example:

    ```
    communities:
    - no-advertise
    ```

  - **settings.addressPools.bgp-advertisements.localpref**
integer

    The `BGP LOCAL_PREF` attribute which is used by BGP best path algorithm.

    Path with higher localpref is preferred over one with lower localpref.

- **settings.addressPools.name**
string

  Required value

  The name of the pool. It should conform to RFC 1123: dot-separated parts in lowercase, consists of alphanumeric characters, ‘-‘. Each part must start and end with an alphanumeric character.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.addressPools.protocol**
string

  Required value

  The protocol used by the speaker to announce services.

  Allowed values: `bgp`, `layer2`

- **settings.bgpCommunities**
Deprecated
object

  Available in editions: Ultimate/EE

  The BGP communities list.

  Example:

```
bgpCommunities:
  no-advertise: 65535:65282
```

- **settings.bgpPeers**
  Deprecated
  array of objects

  Available in editions: Ultimate/EE

  A list of external BGP routers to use with the module.

  Format — a data array similar to that of MetalLB's.

  Default: `[]`

  - **settings.bgpPeers.hold-time**
    integer or string

    The timeout after which the neighboring BGP peer is considered dead. This value is divided by three to get the keep-alive interval.

    The recommended value is `3s` (i.e., keep-alive packets are sent once per second). Note that the BGP protocol does not support values lower than this. By default, the parameter is set to `90s` (i.e., keep-alive packets are sent every 30 seconds).

    Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

  - **settings.bgpPeers.my-asn**
    integer

    Required value

    The AS number in the cluster.

    Allowed values: `0 <= X <= 4294967295`

  - **settings.bgpPeers.node-selector**
    object

    The additional pseudo-selector implemented by the speaker application. It selects nodes that are allowed to connect to external BGP routers. Do not confuse it with `speaker.nodeSelector` and `nodeSelector`.

    An optional parameter.

    The format is `matchLabels` or `matchExpressions`.

    - **settings.bgpPeers.node-selector.matchExpressions**
      array of objects
      - **settings.bgpPeers.node-selector.matchExpressions.key**
        string
      - **settings.bgpPeers.node-selector.matchExpressions.operator**
        string
      - **settings.bgpPeers.node-selector.matchExpressions.values**
        array of strings
    - **settings.bgpPeers.node-selector.matchLabels**
      object
  - **settings.bgpPeers.password**
    string

    Authentication password for BGP-routers enforcing TCP MD5 authenticated sessions.

  - **settings.bgpPeers.peer-address**
    string

    Required value

    The IP address of the external BGP router.

    Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

  - **settings.bgpPeers.peer-asn**
    integer

    Required value

    The AS number on the external BGP router.

    Allowed values: `0 <= X <= 4294967295`

        - **settings.bgpPeers.peer-port**
          integer

          Port to dial when establishing the session.

          Default: `179`

          Allowed values: `0 <= X <= 16384`

        - **settings.bgpPeers.router-id**
          string

          BGP router ID to advertise to the peer.

        - **settings.bgpPeers.source-address**
          string

          The source IP address for outbound connections.

          Pattern: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$`

    - **settings.loadBalancerClass**
      string

      An optional field describing the LoadBalancer class. The LoadBalancerClass field should be used in L2 LoadBalancer mode to specify the MetalLoadBalancerClass that defines the balancer parameters for the Service.

    - **settings.nodeSelector**
      object

      A selector for the main controller. It is the same as the Pod's `spec.nodeSelector` parameter in Kubernetes.

      If the parameter is omitted or `false`, it will be determined automatically.

    - **settings.speaker**
      Deprecated
      object

      Settings for the `speaker` component that implements the LoadBalancer's IPs publishing protocol for external infrastructure.

        - **settings.speaker.nodeSelector**
          Deprecated
          object

          A selector for the speaker DaemonSet. It is the same as the Pod's `spec.nodeSelector` parameter in Kubernetes.

          If the parameter is omitted or `false`, it will be determined automatically.

        - **settings.speaker.tolerations**
          array of objects

          Tolerations for the speaker DaemonSet. They are the same as the Pod's `spec.tolerations` parameter in Kubernetes.

          If the parameter is omitted or `false`, it will be determined automatically.

            - **settings.speaker.tolerations.effect**
              string
            - **settings.speaker.tolerations.key**
              string
            - **settings.speaker.tolerations.operator**
              string
            - **settings.speaker.tolerations.tolerationSeconds**
              integer
            - **settings.speaker.tolerations.value**
              string
    - **settings.tolerations**
      array of objects

      Tolerations for the main controller. They are the same as the Pod's `spec.tolerations` parameter in Kubernetes.

      If the parameter is omitted or `false`, it will be determined automatically.

        - **settings.tolerations.effect**
          string
        - **settings.tolerations.key**

string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

MetalLoadBalancerBGPPeer

Short names: mlbbp

Scope: Cluster
Version: v1alpha1

Describes the settings of a BGP router (neighbor) and the parameters for establishing a session with it.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object
  - **spec.bfd**
    object

    BFD profile configuration. Note: BFD is currently not supported as it requires the FRR BGP implementation, which is not yet available in Deckhouse MetalLB. This field is a placeholder for future use.

    - **spec.bfd.detectMultiplier**
      integer

      The number of missed BFD packets to consider a session down.

    - **spec.bfd.echoInterval**
      integer

      The minimum interval (in milliseconds) between transmitted BFD echo packets.

    - **spec.bfd.echoMode**
      boolean

      Enables or disables BFD echo mode.

    - **spec.bfd.minimumTTL**
      integer

      For multi hop sessions only: configure the minimum expected TTL for an incoming BFD control packet.

      The TTL is measured in network hops.

    - **spec.bfd.passiveMode**
      boolean

Enables passive mode for BFD (does not attempt to bring up the session first).

- **spec.bfd.receiveInterval**
  integer

  The minimum interval (in milliseconds) between received BFD control packets.

- **spec.bfd.transmitInterval**
  integer

  The minimum interval (in milliseconds) between transmitted BFD control packets.

- **spec.holdTime**
  string

  The requested BGP hold time.

  Pattern: `^([0-9]+(ms|s|m|h))+$`

- **spec.myASN**
  integer

  Required value

  The local Autonomous System Number (ASN) used for peering.

  Allowed values: `1 <= X <= 4294967295`

- **spec.passwordSecretRef**
  object

  Reference to the Secret containing the password for BGP session authentication.

  Important: the target Secret MUST have the label `network.deckhouse.io/metallb-bgp-password: "true"`.

  - **spec.passwordSecretRef.name**
    string

    Required value

    Name of the Secret.

  - **spec.passwordSecretRef.namespace**
    string

    Required value

    Namespace of the Secret.

- **spec.peerASN**
  integer

  Required value

  The Autonomous System Number (ASN) of the BGP peer.

  Allowed values: `1 <= X <= 4294967295`

- **spec.peerAddress**
  string

  Required value

  The IP address of the BGP peer.

- **spec.peerPort**
  integer

  The port of the BGP peer.

  Allowed values: `1 <= X <= 65535`

- **spec.routerID**
  string

  BGP router ID to advertise to the peer. Must be a valid IPv4 address.

- **spec.sourceAddresses**
  array of objects

  Per-node override of the local IP address used for peering.

        - **spec.sourceAddresses.address**
          string

          Required value

          IP address to peer from this node.

        - **spec.sourceAddresses.nodeName**
          string

          Required value

          Name of the node.

- **status**
  object

MetalLoadBalancerClass

Short names: mlbc

Scope: Cluster
Version: v1alpha1

MetalLoadBalancerClass is the Schema for the metalloadbalancerclasses API

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object
  - **spec.addressPool**
    array of strings

    An array of public IP address ranges. Each range can be specified using a CIDR notation or by directly specifying the range, for example, `10.10.10.1-10.10.10.100`.

    Examples:

    ```
    addressPool: 10.42.0.0/24
    ```

    ```
    addressPool: 10.33.5.100-10.33.5.106
    ```

  - **spec.isDefault**
    boolean

    Whether MetalLoadBalancerClass is selected for all services by default.

    Example:

    ```
    isDefault: true
    ```

  - **spec.l2**
    object

    Available in editions: SE, SE+, Ultimate/EE

    - **spec.l2.interfaces**
      array of strings

      Available in editions: SE, SE+, Ultimate/EE

      A list of network interfaces from which the assigned IP addresses will be announced. If this field is not filled in or an empty array is specified, the announcement will occur from all interfaces on the node. It is highly recommended to configure this parameter to prevent the announcement of addresses from all node interfaces.

      Example:

      ```
      interfaces:
      - eth1
      - eth2
      ```

    - **spec.nodeSelector**
      object

      Allows restricting the list of nodes from which the announcement will occur. Specified as a set of labels for filtering nodes.

    - **spec.type**
      string

      Selecting the load balancer operating mode.

      Example:

      ```
      type: L2
      ```

- **status**
  object

MetalLoadBalancerConfiguration

Short names: mlbconfig

Scope: Cluster
Version: v1alpha1

Defines the routing topology: from which nodes, to which BGP peers, and which IP address pools to advertise.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object
    - **spec.advertisements**
      array of objects

      List of advertisements and their configurations.

        - **spec.advertisements.bgp**
          object

          BGP-specific advertisement configuration.

            - **spec.advertisements.bgp.aggregationLength**
              integer

              Subnet mask for route aggregation.

              Allowed values: `0 <= X <= 128`

            - **spec.advertisements.bgp.communities**
              array of strings

              BGP communities to associate with the advertisement.

                - **Element of the array**
                  string

                  Pattern: `^([0-9]+:[0-9]+|no-export|no-advertise|local-as|none)$`

            - **spec.advertisements.bgp.localPref**
              integer

              BGP LOCAL_PREF attribute.

              Allowed values: `0 <= X <= 4294967295`

        - **spec.advertisements.poolNames**
          array of strings

          Required value

          List of MetalLoadBalancerPool names to advertise.

    - **spec.bgp**
      object

BGP specific configuration.

- **spec.bgp.peerNames**
  array of strings

  List of MetalLoadBalancerBGPPeer names to associate with this configuration.

- **spec.mode**
  string

  Required value

  The routing mode. Only BGP is supported in this configuration.

  Allowed values: `BGP`

- **spec.nodeSelector**
  object

  Allows limiting the list of nodes from which addresses will be announced.

  Specified as a set of labels for node filtering.

- **status**
  object

MetalLoadBalancerPool

Short names: mlbp

Scope: Cluster
Version: v1alpha1

Describes an IP address pool (MetalLoadBalancerPool) to be used by the load balancer in BGP mode.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object
  - **spec.addresses**
    array of strings

    Required value

    An array of public IP address ranges.

    Each range can be specified using a CIDR notation or by directly specifying the range, for example, `10.10.10.1-10.10.10.100`.

  - **spec.autoAssign**
    boolean

    AutoAssign flag used to prevent MetalLB from automatic allocation for a pool.

    Default: `true`

  - **spec.avoidBuggyIPs**
    boolean

    AvoidBuggyIPs prevents addresses ending with `.0` and `.255` from being used by a pool.

    Default: `false`

- **status**
  object

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> **i** The module lifecycle stage: General Availability

Use the `metallb` module to expose LoadBalancer Services in bare-metal clusters without a cloud load balancer. The module supports L2 and BGP modes.

Example of metallb usage in L2 LoadBalancer mode

Enable the `metallb` module.

Prepare the application to publish:

```
d8 k create deploy nginx --image=nginx
```

Create a MetalLoadBalancerClass:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  l2:
    # Network interfaces that announce the assigned IP addresses.
    interfaces:
      - eth1
  nodeSelector:
    # Selects nodes that run the L2 load balancer.
    node-role.kubernetes.io/loadbalancer: ""
  type: L2
```

In this example, `spec.nodeSelector` selects nodes with the `node-role.kubernetes.io/loadbalancer` label. Make sure the nodes intended to announce the Service IP addresses have this label.

> **⚠** If `spec.l2.interfaces` is omitted or empty, the assigned IP addresses are announced from all network interfaces of the selected nodes. Specify the interfaces intended for address announcements.

Create a Service with the annotation and `loadBalancerClass` set:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  # Selects the MetalLoadBalancerClass named ingress.
  # Can be omitted if the cluster has a MetalLoadBalancerClass with isDefault: true
  # and BGP LoadBalancer mode is not used.
  loadBalancerClass: ingress
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

As a result, the Service of type `LoadBalancer` receives the specified number of IP addresses:

```
d8 k get svc
```

Output example:

```
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)          AGE
nginx-deployment        LoadBalancer   10.222.130.11
192.168.2.100,192.168.2.101,192.168.2.102   8000:30544/TCP   11s
```

When creating a Service, you can also specify certain IP addresses from the pool that will be assigned to it.

You can use the assigned `EXTERNAL-IP` addresses in DNS A records for the application domain. Check that the application responds at these addresses:

```
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.100:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.101:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.102:8000
200
```

Running speakers on nodes with custom taints

In L2 LoadBalancer mode, the speaker runs as a DaemonSet on the nodes selected by the `nodeSelector` parameter of MetalLoadBalancerClass. The speaker tolerates only the following taints known to Deckhouse Platform (DP):

- `node-role.kubernetes.io/master`
- `node-role.kubernetes.io/control-plane`
- `node.deckhouse.io/etcd-arbiter`
- `dedicated.deckhouse.io` and `dedicated` with any value
- `DeletionCandidateOfClusterAutoscaler` and `ToBeDeletedByClusterAutoscaler`

If a selected node has a taint with any other key, the speaker is not scheduled on it, and the addresses are not announced from that node.

To run the speaker on such nodes, add the taint key to the `customTolerationKeys` parameter of the `global` ModuleConfig. The key must match the taint key set on the nodes, for example, in the `spec.nodeTemplate.taints` parameter of NodeGroup. Adding a key there allows both the speaker and the other system components (CNI, CSI, monitoring) to run on such nodes:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 3
  settings:
    modules:
      placement:
        customTolerationKeys:
          - dedicated.example.com
```

Check that the speaker pods are running on the tainted nodes:

```
d8 k -n d8-metallb get pods -l app=l2lb-speaker -o wide
```

The speaker in BGP LoadBalancer mode has the same tolerations, so this setting applies to it as well.

Example of metallb usage in BGP LoadBalancer mode

Enable the `metallb` module and create the following objects for BGP mode:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerPool
metadata:
  name: mypool
spec:
  addresses:
    - 192.168.219.100-192.168.219.200
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerBGPPeer
metadata:
  name: myrouter
spec:
  peerAddress: 172.18.18.10
  peerASN: 64601
  myASN: 64600
  holdTime: 3s
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerConfiguration
metadata:
  name: defaults
spec:
  mode: BGP
  nodeSelector:
    node-role.deckhouse.io/metallb: ""
  bgp:
    peerNames:
      - myrouter
  advertisements:
    - poolNames:
        - mypool
```

In this example, `spec.nodeSelector` selects nodes with the `node-role.deckhouse.io/metallb` label. Make sure the nodes intended to run the speaker pods have this label.

Configure BGP peering on the network equipment.

Example of using L2 LoadBalancer and BGP LoadBalancer modes simultaneously

The module can serve services in L2 LoadBalancer mode and BGP LoadBalancer mode at the same time. The modes are served by different sets of components: `l2lb-controller` and `l2lb-speaker` in L2 LoadBalancer mode, `controller` and `speaker` in BGP LoadBalancer mode. The sets run independently, but they process the same services and see each other’s address pools, so the modes can work together only in the configuration described below.

Enable the module and configure both modes:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  # A default class is not allowed.
  isDefault: false
  l2:
    # Network interfaces that announce the assigned IP addresses.
    interfaces:
      - eth1
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: ""
  type: L2
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerPool
metadata:
  name: mypool
spec:
  # The addresses do not overlap with the MetalLoadBalancerClass pool.
  addresses:
    - 192.168.219.100-192.168.219.200
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerBGPPeer
metadata:
  name: myrouter
spec:
  peerAddress: 172.18.18.10
  peerASN: 64601
  myASN: 64600
  holdTime: 3s
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerConfiguration
metadata:
  name: defaults
spec:
  mode: BGP
  nodeSelector:
    node-role.deckhouse.io/metallb: ""
  bgp:
    peerNames:
      - myrouter
  advertisements:
    - poolNames:
        - mypool
```

Create an L2 LoadBalancer service with MetalLoadBalancerClass specified explicitly:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-l2
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "2"
spec:
  type: LoadBalancer
  # MetalLoadBalancerClass name.
  loadBalancerClass: ingress
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

Create a BGP LoadBalancer service without `spec.loadBalancerClass` and with the pool specified explicitly:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-bgp
  annotations:
    # MetalLoadBalancerPool name.
    network.deckhouse.io/load-balancer-pool: mypool
spec:
  type: LoadBalancer
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

As a result, the `nginx-l2` service will be assigned two addresses from the MetalLoadBalancerClass pool, and the `nginx-bgp` service will be assigned one address from MetalLoadBalancerPool:

```
d8 k get svc
```

Output example:

```
NAME        TYPE           CLUSTER-IP      EXTERNAL-IP                   PORT(S)          AGE
nginx-bgp   LoadBalancer   10.222.41.120   192.168.219.100               8000:31942/TCP   12s
nginx-l2    LoadBalancer   10.222.130.11   192.168.2.100,192.168.2.101   8000:30544/TCP   15s
```

Limitations when running the modes simultaneously

When both modes run in the cluster, the following limitations apply:

- Each L2 LoadBalancer service must specify MetalLoadBalancerClass explicitly in `spec.loadBalancerClass`. The BGP LoadBalancer controller processes services without this parameter, and a service with multiple addresses stops receiving traffic until it is recreated (although the nodes keep responding to ARP requests for its addresses).
- No MetalLoadBalancerClass may have the `isDefault: true` parameter set. Otherwise, the L2 LoadBalancer controller allocates a hidden address from its pool for each BGP LoadBalancer service, and the L2 LoadBalancer pool is silently exhausted.
- A BGP LoadBalancer service must not specify `spec.loadBalancerClass` and is assigned only one IPv4 address. Services with multiple addresses are supported only in L2 LoadBalancer mode.
- Specify a pool for a BGP LoadBalancer service with the `network.deckhouse.io/load-balancer-pool` annotation or an address with the `network.deckhouse.io/load-balancer-ips` annotation. The BGP LoadBalancer controller also sees the MetalLoadBalancerClass pools, so without an annotation it may assign the service an address from the L2 LoadBalancer pool that is not announced via BGP.
- Addresses in MetalLoadBalancerPool must not overlap with addresses in the `addressPool` parameter of MetalLoadBalancerClass. If they overlap, the `speaker` components do not apply the configuration and report the `CIDR ... overlaps with already defined CIDR` error.
- A service can be moved from L2 LoadBalancer mode to BGP LoadBalancer mode only by following the

migration procedure.
- The controller logs may contain messages like `PoolStatusController ... the object has been modified`. They do not affect traffic.

Example of migration from L2 LoadBalancer mode to BGP LoadBalancer mode

During the migration, the L2 LoadBalancer components and the BGP LoadBalancer components run in the cluster simultaneously, and the limitations of simultaneous operation apply. Both controllers can process the same services:

- The BGP LoadBalancer controller processes all services of the `LoadBalancer` type that have no `spec.loadBalancerClass` parameter set.
- The L2 LoadBalancer controller processes services whose `spec.loadBalancerClass` is the name of a MetalLoadBalancerClass. If the cluster has a MetalLoadBalancerClass with the `isDefault: true` parameter, the L2 LoadBalancer controller also processes all other services of the `LoadBalancer` type.

The examples below use the `l2-default` default class. The module creates a class with this name when it converts legacy module settings. Your cluster may have a default class with a different name. Find it and use its name instead of `l2-default`:

```
d8 k get metalloadbalancerclass -o custom-columns='NAME:.metadata.name,DEFAULT:.spec.isDefault'
```

> ⚠ The procedure described below is the only supported way to migrate from L2 LoadBalancer mode to BGP LoadBalancer mode. Enabling BGP LoadBalancer mode without preparing L2 services, having an address in the pools of both modes at the same time, and manually creating MetalLB objects (IPAddressPool, BGPAdvertisement, etc.) lead to service unavailability.
>
> The migration involves a brief unavailability: less than 1 second when recreating a service in step 1 and about 11 seconds for each service moved to BGP LoadBalancer mode in step 4.

Before you begin

Make sure that the cluster and the network meet the following requirements:

- The cluster runs the Enterprise, Ultimate, or Certified Security Pro edition. BGP LoadBalancer mode is not available in other editions.
- The network equipment accepts `/32` BGP announcements from the nodes for addresses from the L2 LoadBalancer subnet. Such a route has a longer prefix, so the router prefers it over the route to the directly connected subnet.
- Clients of the moved services are outside the L2 LoadBalancer subnet, or the network sends their traffic to these addresses through the router. Clients in the same subnet resolve the service address via ARP and do not use the router. After the move, the nodes no longer respond to ARP requests for the address, and such clients lose access to the service.

Step 1. Binding L2 services to MetalLoadBalancerClass

If you enable BGP LoadBalancer mode while L2 services have no `spec.loadBalancerClass` set, the BGP LoadBalancer controller starts processing them. Services with multiple addresses (the `network.deckhouse.io/l2-load-balancer-external-ips-count` annotation is greater than `1`) stop receiving traffic: the controller cannot assign multiple IPv4 addresses to a service and clears its status, while the nodes keep responding to ARP requests for these addresses. Disabling BGP LoadBalancer mode does not restore such services, only recreating them helps.

Find services of the `LoadBalancer` type that have no `spec.loadBalancerClass` set (`<none>` in the `CLASS` column):

```
d8 k get svc -A -o custom-
columns='NAMESPACE:.metadata.namespace,NAME:.metadata.name,TYPE:.spec.type,CLASS:.spec.lo
adBalancerClass,EXTERNAL-IP:.status.loadBalancer.ingress[*].ip' | awk 'NR==1 ||
$3=="LoadBalancer"'
```

Output example:

```
NAMESPACE   NAME               TYPE           CLASS    EXTERNAL-IP
default     nginx-deployment   LoadBalancer   <none>   192.168.2.100,192.168.2.101
default     web                LoadBalancer   <none>   192.168.2.120
```

Recreate each service that should remain in L2 LoadBalancer mode with `spec.loadBalancerClass` set explicitly. This parameter cannot be changed for an existing service. To keep the current addresses of the service, list them in the `network.deckhouse.io/load-balancer-ips` annotation.

Save the service manifest to the `nginx-deployment.yaml` file:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  namespace: default
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "2"
    network.deckhouse.io/load-balancer-ips: "192.168.2.100,192.168.2.101"
spec:
  type: LoadBalancer
  # MetalLoadBalancerClass name.
  loadBalancerClass: l2-default
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

Delete the service and create it again from the saved manifest:

```
d8 k -n default delete svc nginx-deployment && d8 k apply -f nginx-deployment.yaml
```

Do not set `spec.loadBalancerClass` for services that will be moved to BGP LoadBalancer mode. It is enough to pin their current address with the `network.deckhouse.io/load-balancer-ips` annotation, no recreation is required:

```
d8 k -n default annotate svc web network.deckhouse.io/load-balancer-ips=192.168.2.120
```

In BGP LoadBalancer mode, a service is assigned only one IPv4 address, so only services with a single address can be moved to this mode. Keep services with multiple addresses in L2 LoadBalancer mode.

Step 2. Disabling the default MetalLoadBalancerClass

While MetalLoadBalancerClass has the `isDefault: true` parameter set, the L2 LoadBalancer controller also processes BGP LoadBalancer services. For each such service, it allocates an address from the MetalLoadBalancerClass pool and announces it via ARP, although this address is not shown in the service status. Such hidden addresses silently consume the L2 LoadBalancer pool and can lead to its exhaustion. Services with an address pinned by the `network.deckhouse.io/load-balancer-ips` annotation are not affected by this issue.

Set the `isDefault` parameter to `false`:

```
d8 k patch metalloadbalancerclass l2-default --type=merge -p '{"spec":{"isDefault":false}}'
```

The parameter affects only new services. Services that have already been bound to the default MetalLoadBalancerClass remain bound to it.

Addresses allocated in L2 LoadBalancer mode are represented by SDNInternalL2LBService objects named `<service name>-<MetalLoadBalancerClass name>-<index>`:

```
d8 k get sdninternall2lbservices -A
```

If BGP LoadBalancer mode has already been enabled earlier and the list contains objects for services of this mode, these services received a hidden address from the L2 LoadBalancer pool before `isDefault` was disabled. Disabling `isDefault` does not release such an address, since the service remains bound to MetalLoadBalancerClass. Specify the current address of such a service from the BGP LoadBalancer pool in its `network.deckhouse.io/load-balancer-ips` annotation. This address is not in the L2 LoadBalancer pool, so the L2 LoadBalancer controller releases the hidden address, and the service address does not change. An example for the `web-bgp` service with the `192.168.219.100` address:

```
d8 k -n default annotate svc web-bgp network.deckhouse.io/load-balancer-ips=192.168.219.100
```

Step 3. Enabling BGP LoadBalancer mode

Create a MetalLoadBalancerPool, a MetalLoadBalancerBGPPeer, and a MetalLoadBalancerConfiguration as shown in the BGP LoadBalancer example. Addresses in MetalLoadBalancerPool must not overlap with addresses in the `addressPool` parameter of MetalLoadBalancerClass.

Make sure that the components of both modes are running in the module:

```
d8 k -n d8-metallb get pods
```

Check that the addresses of L2 services have not changed:

```
d8 k get svc -A
```

While both modes are running simultaneously, the controller logs may contain messages like `PoolStatusController ... "l2-default": the object has been modified`. They do not affect traffic.

Step 4. Moving a single-address service to BGP LoadBalancer mode

Move services one at a time. With a single `d8 k apply` command, remove the service address from the `addressPool` parameter of MetalLoadBalancerClass and add it to the `addresses` parameter of MetalLoadBalancerPool. The address must not be in the pools of both modes at the same time: in this case, the `speaker` components cannot apply the configuration due to overlapping pools (`CIDR "192.168.2.120/32" in pool "l2-default" overlaps with already defined CIDR`).

Prepare a file with the MetalLoadBalancerClass and MetalLoadBalancerPool manifests. Take the current parameters of both objects from the cluster and change only the address lists:

```
d8 k get metalloadbalancerclass l2-default -o yaml
d8 k get metalloadbalancerpool mypool -o yaml
```

Put MetalLoadBalancerClass first in the file. `d8 k apply` applies the manifests in the order they appear, so the address is removed from the L2 LoadBalancer pool before it is added to MetalLoadBalancerPool.

An example of moving the `192.168.2.120` address that belongs to the `192.168.2.100-192.168.2.150` range. The `migrate-192.168.2.120.yaml` file:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: l2-default
spec:
  addressPool:
    - 192.168.2.100-192.168.2.119
    - 192.168.2.121-192.168.2.150
  isDefault: false
  # Keep the current values of the parameters.
  l2:
    interfaces:
      - eth1
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: ""
  type: L2
---
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerPool
metadata:
  name: mypool
spec:
  addresses:
    - 192.168.219.100-192.168.219.200
    - 192.168.2.120/32
```

Apply both manifests with a single command:

```
d8 k apply -f migrate-192.168.2.120.yaml
```

The L2 LoadBalancer controller releases the address, after which the BGP LoadBalancer controller assigns it to the service and starts announcing it via BGP. Check the service events:

```
d8 k -n default describe svc web
```

The service events should show, in order, `ClearAssignment` (the address is removed from the L2 LoadBalancer pool), `IPAllocated` (the address is assigned by the BGP LoadBalancer controller), and `nodeAssigned` with `protocol "bgp"` (the address is announced via BGP).

Make sure that the router has a `/32` route to the service address via the cluster nodes, and that the nodes no longer respond to ARP requests for this address.

Additional configuration examples for Service

After setting up a load balancer, use Service annotations to control address allocation and sharing.

Creating a Service and assigning it specific IP addresses from the pool

To assign specific addresses to a Service, list them in the `network.deckhouse.io/load-balancer-ips` annotation. To assign more than one address, also set the `network.deckhouse.io/l2-load-balancer-external-ips-count` annotation to the number of addresses to allocate from the pool. The value must not be less than the number of addresses in `network.deckhouse.io/load-balancer-ips`; otherwise, the extra addresses are not assigned.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # The number of addresses that will be allocated from the pool declared in
MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # A list of addresses from the pool declared in MetalLoadBalancerClass that will be
allocated to the Service.
    network.deckhouse.io/load-balancer-ips: "192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  # Selects the MetalLoadBalancerClass named ingress.
  # Can be omitted if the cluster has a MetalLoadBalancerClass with isDefault: true
  # and BGP LoadBalancer mode is not used.
  loadBalancerClass: ingress
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
```

As a result, the Service of type `LoadBalancer` receives the specified IP addresses:

```
d8 k get svc
```

Output example:

```
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)          AGE
nginx-deployment        LoadBalancer   10.222.130.11
192.168.2.102,192.168.2.103,192.168.2.104   8000:30544/TCP   11s
```

Creating a Service with a specific IP address

To assign a specific IP address to a Service, add the `network.deckhouse.io/load-balancer-ips` annotation:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    network.deckhouse.io/load-balancer-ips: 192.168.2.105
spec:
  loadBalancerClass: ingress
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Creating Services with shared IP addresses

To allow multiple Services to share an IP address, add the `network.deckhouse.io/load-balancer-shared-ip-key` annotation to each Service. Its value is the sharing key. Services can share an IP address under the following conditions:

- They have the same sharing key.
- Their protocol and port combinations do not conflict (for example, TCP/53 and UDP/53 can use the same IP address).
- They use `externalTrafficPolicy: Cluster`, or they both point to the exact same set of pods (that is, the pod selectors are identical).

> ⚠ Even when these conditions are met, MetalLB does not guarantee that the Services will receive the same IP address. To request a specific shared address, set `network.deckhouse.io/load-balancer-ips` on both Services.

The following example configures two Services to share an IP address:

```
apiVersion: v1
kind: Service
metadata:
  name: dns-service-tcp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress
  ports:
    - name: dnstcp
      protocol: TCP
      port: 53
      targetPort: 53
  selector:
    app: dns
---
apiVersion: v1
kind: Service
metadata:
  name: dns-service-udp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress
  ports:
    - name: dnsudp
      protocol: UDP
      port: 53
      targetPort: 53
  selector:
    app: dns
```

Assigning an IP address pool to a Service in BGP LoadBalancer mode

In BGP LoadBalancer mode, assign an IP address pool to a Service using the `network.deckhouse.io/load-balancer-pool` annotation. For L2 LoadBalancer mode, use the MetalLoadBalancerClass settings, as shown in the L2 LoadBalancer mode example.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    network.deckhouse.io/load-balancer-pool: mypool
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module multitenancy-manager

> **i** Included in extensions: **Multitenancy**

> ⚠ The module lifecycle stage: Preview

The module enables the creation of projects in a Kubernetes cluster. **Project** is an isolated environment where applications can be deployed.

Main features

- Creating projects from project templates. A template sets the pod security profile, network isolation, log collection, node placement and other policies of the project namespaces.
- Several namespaces in one project. The template settings and the access granted to the project apply to all of them.
- Access to the whole project through `.spec.administrators`, ProjectRoleBinding and ClusterProjectRoleBinding.
- A resource quota for each project.
- A project for every namespace: a namespace created directly becomes a project automatically.
- Control over which cluster-wide resources, such as StorageClass or ClusterIssuer, projects can use.

> ⚠ The secret-copier module cannot be used together with the `multitenancy-manager` module: it copies Secrets into every namespace, project namespaces included.

Project composition

A project includes:

- **The main namespace** — created automatically, its name matches the project name.
- **Additional namespaces** (optional) — created with the ProjectNamespace resource and named `<project name>-<name>`. The project settings (security policies, network policies, log collection, etc.) and the access granted to the project automatically apply to all of its namespaces.
- **Standard fields** — the list of administrators (`.spec.administrators`) and resource quotas (`.spec.quota`), managed by the Project resource itself independently of the template.
- **Resources from the template** (optional) — policies and settings described in a ProjectTemplate and created by the controller in the project namespaces, and the objects of its `manifests`, applied with the rights of a project administrator.

How the module works

This section describes how the module creates projects, adds namespaces to them, grants access and turns existing namespaces into projects.

Creating a project

To create projects, the following Custom Resources are used:

- ProjectTemplate — a resource that describes the project template: which settings and policies its namespaces get and which parameters can be passed when creating a project;
- Project — a resource that describes a specific project.

When creating a Project resource from a specific ProjectTemplate, the following happens:

1. The parameters passed are validated against the OpenAPI specification (the `parametersSchema.openAPIV3Schema` field of ProjectTemplate);
2. A namespace is created with the name of the Project;
3. The project resources are created from the template:
   - in modern templates (`deckhouse.io/v1alpha2`) the settings, such as the pod security profile, network isolation, log collection and node placement, are described by structured fields. The controller itself creates the corresponding objects in every namespace of the project. Other objects go to `manifests` and are applied with the rights of a project administrator;
   - a template with a Helm `resourcesTemplate` is rendered through Helm. This form is deprecated, and so is the `deckhouse.io/v1alpha1` version of ProjectTemplate, which has no other form (see details). No template can get a new `resourcesTemplate`, a template is created through `deckhouse.io/v1alpha2` only, and the field goes away in a future release, so move such a template to structured fields and `manifests`;
4. The standard fields of the project are applied independently of the template: `.spec.quota` is reconciled into a ProjectQuotaStatus of the project, which puts the `d8-project-quota` ResourceQuota

into every namespace of the project, and `.spec.administrators` into an auto-managed ProjectRoleBinding.

The Project API is served as `deckhouse.io/v1alpha3`. `v1alpha2` manifests keep working through a conversion webhook, and `v1alpha1` is no longer served (see Project API versions). The `projectTemplateName` field is optional: when omitted, the `simple` template is used, which only creates the project namespace.

Project names are validated on creation. Names longer than 61 characters, names with the system prefixes `d8-` and `kube-`, names of existing namespaces or of namespaces that a ProjectNamespace has claimed, and names of namespaces whose ServiceAccounts a cluster-wide binding or a binding of another project still names are not allowed. A project `foo-bar` beside a project `foo` is allowed, but it is a separate project and not an additional namespace of `foo`. The webhook warns about it, and an alert lists such pairs (see details).

⚠ A change of a project template applies to every project created from it.

Multiple namespaces in a project

A project can span several namespaces. Additional namespaces are created with the ProjectNamespace resource in the main namespace of the project and are named `<project name>-<name>` (for example, `backend-cache` for the `backend` project).

The following automatically applies to the additional namespaces:

- the project template settings — the pod security profile, network policies, log collection, and other policies are created in every namespace of the project;
- the access granted via ProjectRoleBinding and ClusterProjectRoleBinding, including the automatic access of the project administrators.

The project quota (`.spec.quota`) applies in every namespace of the project as one limit on the sum of their usage.

Deleting a project deletes all of its namespaces — both the main and the additional ones. See the usage examples for details.

Access to a project

User access to a project is managed at the level of the whole project rather than individual namespaces:

- The subjects listed in `.spec.administrators` automatically get the project administrator role (`d8:project:admin`).
- The ProjectRoleBinding resource grants a role in all namespaces of one project.
- A ClusterProjectRoleBinding grants a role in all projects of the cluster at once, except the virtual ones.

A ProjectRoleBinding or ClusterProjectRoleBinding can grant only ClusterRoles named `d8:project:*`, `d8:namespace:*`, `d8:project-capability:*`, `d8:namespace-capability:*` or `d8:custom:*`. For details on the roles, see the user-authz module documentation. See the usage examples for details on the bindings.

Automatic project creation for namespaces

Users can still create namespaces directly (`d8 k create namespace`), unless the `allowNamespacesWithoutProjects` parameter is `false`: then users create environments as Project objects, and only cluster administrators (`system:masters`, `system:sudouser`), Deckhouse, the module controller and system components create namespaces directly. Every namespace that belongs to no project becomes a project of its own, with the same name and a full-fledged template:

- the template is picked from what the namespace already carries — `secure` if it has vulnerability scanning enabled, `default` if it has a Pod Security Standard or extended monitoring, and `simple` (namespace only) otherwise;
- the project parameters are filled in from the current state of the namespace, so adopting it changes nothing inside;
- from then on the project is the source of truth and is edited like any other. The namespace cannot be deleted directly: to remove the environment, delete the Project, and its namespace is deleted with it.

System namespaces (`d8-*`, `kube-*`, `upmeter-*`, `default`, and anything labeled `heritage: deckhouse` or `heritage: upmeter`) are never adopted: they are listed on the virtual `deckhouse` project (except `default`). No label leaves a new user namespace without a project: users cannot put the `heritage: upmeter` label on a namespace, and only the module sets the `projects.deckhouse.io/project` label. A namespace whose name is longer than 61 characters (the Project name limit) is also skipped, and so is a namespace named `deckhouse`: a Project of that name is the virtual project.

Users cannot create a namespace that could not become a project (see details).

Isolating a project

A namespace alone does not isolate a project. The project adds the following:

- Access to the whole project through `.spec.administrators`, ProjectRoleBinding and ClusterProjectRoleBinding. A plain RoleBinding in one namespace is allowed for delegatable roles.
- A resource limit on the whole project through `.spec.quota`, applied as the `d8-project-quota` ResourceQuota in every namespace of the project and as a limit on the sum of their usage.
- Network isolation through the `networkPolicy.mode: Isolated` field of the template.
- The pod security profile through the `podSecurityStandard` field of the template.

Managing access to cluster-wide resources

Objects in project namespaces can reference cluster-wide resources. For example, a PersistentVolumeClaim can use a StorageClass, a Certificate can use a ClusterIssuer, and a RoleBinding can use a ClusterRole. The module allows cluster administrators to define which cluster-wide resources can be used from project namespaces and which values are used by default.

This mechanism works independently of RBAC. RBAC determines *who can create and modify* objects, while the cluster-wide resource access mechanism determines *which resources* those objects can use.

How access management works

The following resources are used to manage access to cluster-wide resources:

- GrantableClusterResourceDefinition registers a type of cluster-wide resource whose access can be managed. These resources are provided by Deckhouse Platform or by module developers.
- GrantableClusterResourceReference defines where a registered cluster-wide resource is used, for example, which resource field contains a reference to it. These resources are provided by modules.
- ClusterResourceGrantPolicy defines access rules. Using labels, a cluster administrator selects the projects to which the policy applies and defines the allowed and denied resources, as well as the resource used by default.
- In every namespace of every project, the controller keeps an AvailableClusterResource for each GrantableClusterResourceDefinition. It lists the cluster-wide resources available to the project under the policies that apply and is read-only.
- When an object is created or modified, webhooks check whether it is allowed to use the specified cluster-wide resource. When an object is created, the resource configured as the project default can also be inserted automatically.

The webhooks reject requests while the module controller is unavailable. Until it is back, objects of the registered kinds (for example, Pod, Deployment, StatefulSet, Job, PersistentVolumeClaim, Service or RoleBinding) cannot be created or changed in project namespaces, except by members of the `system:masters` and `system:nodes` groups and by the service accounts of the `d8-system`, `kube-system` and `d8-user-authz` namespaces. This also applies to updates that controllers make, such as label, annotation or finalizer patches of pods and workloads, including controllers of Deckhouse Platform modules running in other namespaces. A request goes through two webhooks, and the API server awaits a reply from each of them for up to 10 seconds. To keep the controller available, enable `highAvailability`.

```
flowchart LR
    A["Module developer or Deckhouse Platform
provides GrantableClusterResourceDefinition
and GrantableClusterResourceReference"] --> C
    B["Cluster administrator
creates
ClusterResourceGrantPolicy"] --> C["Controller"]
    C --> D["Creates AvailableClusterResource
in each project namespace"]
    E["User creates an object
(for example,
PersistentVolumeClaim)"] --> F["Mutating webhook
/defaults"]
    F --> G["Validating webhook
/is-granted"]
    D -. Available names .-> G
    G --> H["Object created
or rejected"]
```

Until the administrator creates a ClusterResourceGrantPolicy, resource availability is determined by their registration: resources are available to all projects if `defaultAvailability: All` (the default value) is set in GrantableClusterResourceDefinition and the resource does not match the `excluded` or `reserved` filters.

Resource quotas are not part of this mechanism. They are managed using the standard Kubernetes ResourceQuota resource.

Access checks are performed only for objects in project namespaces. If an access policy changes, cluster-wide resources already used by existing objects remain available to those objects.

Access checks and default value assignment

The mechanism determines cluster-wide resource availability and default values based on registered resources and policies that apply to the project.

Determining cluster-wide resource availability

If multiple rules apply to a cluster-wide resource, its availability is determined in the following order:

1. The `excluded` value in GrantableClusterResourceDefinition: the resource is unavailable regardless of policy settings.
2. The `denied` and `deniedSelector` values in the corresponding ClusterResourceGrantPolicy entry.
3. The `allowed` and `allowedSelector` values in the corresponding ClusterResourceGrantPolicy entry.
4. The `reserved` value in GrantableClusterResourceDefinition: a reserved resource that the previous steps have not made available is unavailable. The next steps do not apply to it, so `availabilityDefault: All` does not make it available.
5. The `availabilityDefault` value in the ClusterResourceGrantPolicy entries that apply to the project. If an entry sets `allowed` or `allowedSelector` without `availabilityDefault`, every resource not on its lists is unavailable. If one applicable entry sets `All`, it wins over `None` in the others.
6. The `defaultAvailability` value in GrantableClusterResourceDefinition.

The first matching step decides. Steps 2, 3 and 5 combine the entries of all ClusterResourceGrantPolicy objects that apply to the project, including the service policies of its template and of the project itself.

Assigning default values

The behavior when an object is created depends on the mode configured in GrantableClusterResourceReference:

- `None`: The value is checked for availability but is not assigned automatically.
- `FillEmpty`: If no value is specified, the project default is assigned.
- `Coerce`: If no value is specified or the specified cluster-wide resource is unavailable to the project, the project default is assigned. Replacing a specified value is reported as an admission warning in the response to the request.

The project default is determined in the following order:

1. The `default` value from the corresponding ClusterResourceGrantPolicy entry.
2. The value determined using `defaultFrom` in GrantableClusterResourceDefinition.
3. If no value is found, no default is assigned.

Checking existing objects

When an access policy changes, values already used by existing objects are preserved. When an object is updated, only new values are checked. If the value of a field has not changed, it remains valid even after access to the resource has been restricted.

This allows existing objects to continue operating after access to cluster-wide resources is restricted.

System requests

Requests from platform components and from members of the `system:masters` group are not checked. The platform components are the API server, the ServiceAccounts of Deckhouse, of this module and of the `user-authz` module, all ServiceAccounts of the `d8-system`, `kube-system` and `d8-user-authz` namespaces, and kubelets. To test a policy, send the request as a project user, for example with `d8 k --as <USER_NAME>`.

Monitoring access policy violations

If an existing object uses a cluster-wide resource that becomes unavailable to the project after a policy change, the object continues to operate, but the discrepancy is detected by the monitoring system.

When such objects are detected, the `ClusterResourceGrantPolicyViolation` alert is triggered. Information about violations is available on the Grafana dashboard under “Security” → “Cluster Resource Grant Violations”.

The `d8_cluster_objects_grant_violated` metric is used for monitoring. It is exported by the module controller, which recounts the violations of a project namespace on every reconcile and at least every two minutes, using the same availability rules as the admission webhook.

To avoid duplication, the same violation is not counted again for child objects. For example, if a Deployment uses an unavailable RuntimeClass, the violation is recorded only for the Deployment, not for its ReplicaSets or Pods.

Resources registered by Deckhouse Platform

Deckhouse Platform registers the following cluster-wide resources:

Definition name	Cluster-wide resource	Where it is used	Default assignment mode
`storageclasses`	StorageClass (storage.k8s.io)	PersistentVolumeClaim `.spec.storageClassName` and the `volume.beta.kubernetes.io/storage-class` annotation. PersistentVolumeClaim templates in StatefulSets and generic ephemeral volumes of Pods and workload controllers	`Coerce` for PersistentVolumeClaims in general. `None` for PersistentVolumeClaims whose `.spec.dataSource` or `.spec.dataSourceRef` references a VolumeSnapshot or a PersistentVolumeClaim, for PersistentVolumeClaims with the beta annotation, and for templates
`loadbalancerclasses`	`loadbalancerclass` value	Service `.spec.loadBalancerClass` (for services of the `LoadBalancer` type)	`FillEmpty`
`clusterissuers`	ClusterIssuer (cert-manager.io)	Certificate `.spec.issuerRef.name`; Ingress: `cert-manager.io/cluster-issuer` annotation	`FillEmpty` or `None`
`clusterroles`	ClusterRole (rbac.authorization.k8s.io)	RoleBinding `.roleRef.name`	`None`
`runtimeclasses`	RuntimeClass (node.k8s.io)	Pod `.spec.runtimeClassName`; the pod template of ReplicationController, PodTemplate, Deployment, StatefulSet, DaemonSet, ReplicaSet, Job and CronJob	`None`
`applicationpackages`	ApplicationPackage (deckhouse.io)	Application `.spec.packageName`	`None`
`priorityclasses`	PriorityClass (scheduling.k8s.io)	Pod `.spec.priorityClassName`; pod template of ReplicationController, Deployment, StatefulSet, DaemonSet, ReplicaSet, Job and CronJob; PodTemplate `.template.spec.priorityClassName`	`None` for Pod and PodTemplate, `FillEmpty` for the pod templates of workloads

The `clusterroles` registration excludes every ClusterRole without the `rbac.deckhouse.io/delegatable: "true"` label. By default, a RoleBinding in a project can reference the `d8:namespace:*` and `d8:project:*` roles, the deprecated `d8:use:role:*` aliases, and the `user-authz:user`, `user-authz:privileged-user`, `user-authz:editor` and `user-authz:admin` roles of the basic model.

The `priorityclasses` registration sorts the priority classes of Deckhouse Platform into three groups:

Classes	Availability to projects
`system-node-critical`, `system-cluster-critical`, `cluster-medium`, `cluster-low`, `standby`	Excluded and reserved for the platform. No ClusterResourceGrantPolicy can open them. In project namespaces, only members of the `system:masters` and `system:nodes` groups and the service accounts of the `d8-system`, `kube-system` and `d8-user-authz` namespaces can create objects that use them

Classes	Availability to projects
`production-high`, `deployment-machinery`	Reserved and unavailable by default. The `production-high` class has a higher priority than the `cluster-medium` platform components, and `deployment-machinery` is meant for the components that build and deploy applications, such as dedicated CI/CD projects
`production-medium`, `production-low`, `staging`, `develop` and the classes added by the administrator	Available by default

To grant a reserved class to trusted projects without restricting the other classes, list it in `allowed` and set `availabilityDefault: All` (for details, refer to “Granting a reserved resource”):

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: priorityclasses-production-high
spec:
  projectSelector:
    matchLabels:
      tier: critical
  resources:
    - resourceName: priorityclasses
      allowed:
        - production-high
      availabilityDefault: All
```

Objects that already use a reserved class keep working and can be updated as long as they keep the class. Creating a new object with the class in a project that is not granted it is rejected.

Classes of priority no lower than `cluster-medium` are by default unavailable to projects, since using them may allow applications to outrank the critical platform components (`production-high`). Also, the classes intended for infrastructure components rather than custom applications are unavailable by default as well (the excluded classes and `deployment-machinery`). Some classes that are available by default have a higher priority than `cluster-low`. Therefore a project with the `production-medium`, `production-low` or `staging` class can preempt the platform components that run with `cluster-low` (for example, Dex, `descheduler`, the `cert-manager` controller, the Istio operator and Kiali, and the etcd backup CronJob) and the pods of other projects. If this is not acceptable, restrict the classes available to projects with a policy, for example, `allowed: [develop]`.

Every namespace outside an existing project is adopted into a project, so the excluded classes apply to infrastructure components that an administrator installs into a namespace of their own as well. Objects created there by a cluster administrator signed in through Dex, by GitOps tools or by third-party operators (for example, CSI or CNI agents with `system-node-critical`) are rejected, and no policy allows them.

Classes that the administrator or a third-party chart adds are available by default whatever their priority. To close a class with a high priority for all projects, list it in `denied` of a policy that selects all projects.

The `default` value of a policy for `priorityclasses` applies only to pod templates (Deployment, StatefulSet, Job and so on) that do not specify a class. If a user or a module controller creates a pod without `.spec.priorityClassName` (for example, the pod for domain ownership verification of `cert-manager`), gets the cluster default class from Kubernetes before the check. In Deckhouse Platform it is `develop`. If this class is unavailable for the project, such a pod is rejected, so when restricting priority classes, keep the cluster default class available. If a controller rewrites the whole pod template on every reconciliation (for example, via Server-Side Apply in GitOps tools), it removes the class inserted by `default` on its next reconciliation. As a result, the corresponding workloads restart once and end up with the cluster default class.

The priority classes of a namespace can also be limited with `priorityClassNames` of an OperationPolicy of the `admission-policy-engine` module.

The `clusterissuers` definition is shipped by the `cert-manager` module, so it is registered only when that module is enabled.

The `storageclasses` definition and its GrantableClusterResourceReference objects (`storageclasses-pvc` and the references for templates) are registered by this module unless the `storage-foundation` module of version 1.0.3 or later is enabled. While such a version of `storage-foundation` is enabled, that module registers them under the same names and can extend them. When `storage-foundation` is enabled, disabled or updated to such a version, the objects pass from one module to the other, so ClusterResourceGrantPolicy objects keep restricting StorageClass.

The `runtimeclasses` registration checks the pod templates of the workloads because the pods they create

are written by the Kubernetes controllers, which the check does not cover. No RuntimeClass gets a value assigned by default: Kubernetes has no default RuntimeClass, and assigning one would change where the pods are scheduled and how much of the quota they use. Deckhouse Platform doesn't create its own RuntimeClasses, so no class is excluded from the check. Classes created by the administrator or shipped by external modules (for example, the `gpu` module) are available to every project until a ClusterResourceGrantPolicy narrows them. A request without `runtimeClassName`, or with an unchanged value, is checked without reading the API server.

With this registration, creating and changing pods in project namespaces depends on the availability of the module: the check rejects the request (`failurePolicy: Fail`) while the module controller is unavailable. Pods created by Kubernetes controllers, by Deckhouse Platform and by `kube-system` are not checked. Pods created by other operators are checked as pods, so a rejection is returned to the operator, not to the user who created its resource. For example, if KubeVirt has `defaultRuntimeClass` set and the project may not use that RuntimeClass, `virt-controller` cannot create the `virt-launcher` Pod, and the virtual machine does not start.

Access management resources

Resource	Scope	Created by	Manual creation	Purpose
GrantableClusterResourceDefinition	Cluster	Module developer or Deckhouse Platform	Allowed for custom resources	Registers a type of cluster-wide resource whose access can be managed
GrantableClusterResourceReference	Cluster	Module developer	Allowed for fields of custom resources	Defines where a registered cluster-wide resource is used
ClusterResourceGrantPolicy	Cluster	Cluster administrator	Required	Defines allowed and denied resources, as well as the resource used by the project by default
AvailableClusterResource	Namespace	Controller (automatically)	Prohibited (protected by a webhook)	Read-only catalog of resources available to the project

For detailed access configuration scenarios, viewing resources available to projects, access checks and default assignment rules, and recommendations for module developers, refer to "Usage examples".

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The multitenancy-manager module: configuration

> Included in extensions: **Multitenancy**

> ⚠ The module lifecycle stage: Preview

The module has 22 alerts.

The module does not have any mandatory parameters.

The module has 22 alerts.

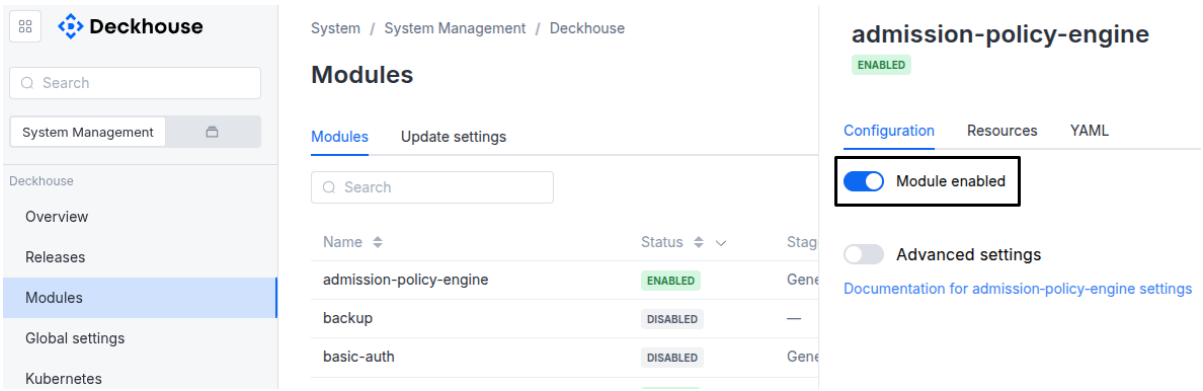
The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules"

section, open the `multitenancy-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable multitenancy-manager
  ```

- **Using ModuleConfig `multitenancy-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `multitenancy-manager` (create it if necessary);

  Example of a manifest to enable module `multitenancy-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: multitenancy-manager
  spec:
    enabled: true
  ```

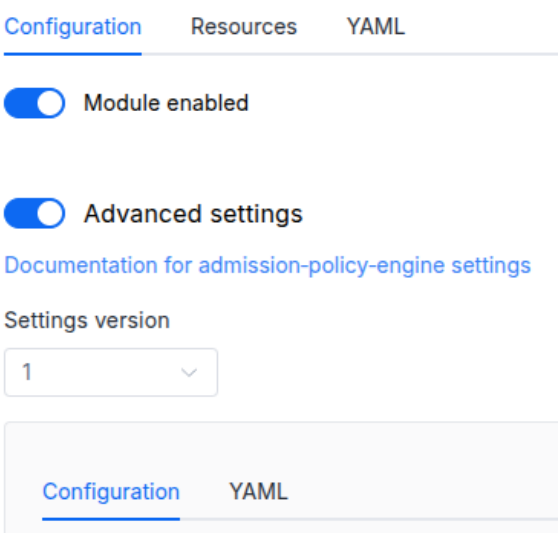
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `multitenancy-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `multitenancy-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `multitenancy-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `multitenancy-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc multitenancy-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `multitenancy-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `multitenancy-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: multitenancy-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.allowNamespacesWithoutProjects**
    boolean

    Whether namespaces can be created directly, outside a Project.

    The parameter decides only who creates namespaces directly. Whatever its value, the controller turns every namespace that is not a system one into a project of its own.

    If `true`, anyone allowed to create namespaces can create one directly (`d8 k create namespace`), and the controller turns such a namespace into a project of its own (see Automatic project creation for namespaces).

    If `false`, users create environments as Project objects, and additional namespaces of a project as ProjectNamespace objects. Only cluster administrators (the `system:masters` group and `system:sudouser`), Deckhouse, the module controller and system components create namespaces directly; the full list is in Who bypasses admission. Anyone can create the `default` namespace. The check is a ValidatingAdmissionPolicy that the API server evaluates itself, so it does not depend on the module controller being available.

    Default: `true`

  - **settings.highAvailability**
    boolean

    Whether the module runs in the high availability mode.

    If not set, the mode is chosen automatically, as described for the global `highAvailability` parameter.

    Examples:

    ```
    highAvailability: true
    ```

```
highAvailability: false
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The multitenancy-manager module: custom resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> ⚠ The module lifecycle stage: Preview

ClusterProjectRoleBinding

Short names: cprb

Scope: Cluster
Version: v1alpha3

ClusterProjectRoleBinding grants a role to subjects across all namespaces of all projects in the cluster, except the virtual projects `deckhouse` and `default`.

- **spec**
  object

  Required value

  - **spec.roleRef**
    object

    Required value

    Reference to the granted ClusterRole. Allowed prefixes: `d8:project:*`, `d8:namespace:*`, `d8:project-capability:*`, `d8:namespace-capability:*`, `d8:custom:*`.

    - **spec.roleRef.kind**
      string

      Required value

      Kind of the granted role.

      Allowed values: `ClusterRole`

    - **spec.roleRef.name**
      string

      Required value

      Name of the granted ClusterRole.

  - **spec.subjects**
    array of objects

    Required value

    Subjects (users, groups or service accounts) the role is granted to.

    - **spec.subjects.apiGroup**
      string

      API group of the subject: `rbac.authorization.k8s.io` for `User` and `Group`, empty for `ServiceAccount`.

    - **spec.subjects.kind**
      string

      Required value

Kind of the subject.

Allowed values: `User`, `Group`, `ServiceAccount`

- **spec.subjects.name**
  string

  Required value

  Name of the subject: the user name (`User.spec.email`) for `User`, `Group.spec.name` for `Group`, the ServiceAccount name for `ServiceAccount`.

- **spec.subjects.namespace**
  string

  Namespace of the ServiceAccount.

  If omitted, the binding names the ServiceAccount with this name in each namespace of every project.

- **status**
  object
  - **status.boundProjects**
    integer

    Number of projects the binding is currently applied to.

  - **status.conditions**
    array of objects

    Current state of the object.

    - **status.conditions.lastProbeTime**
      string

      Time of the last check of the condition.

    - **status.conditions.lastTransitionTime**
      string

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Message with details of the condition state.

    - **status.conditions.status**
      string

      Condition state: `True`, `False` or `Unknown`.

    - **status.conditions.type**
      string

      Condition type.

  - **status.observedGeneration**
    integer

    Last generation of the object processed by the controller.

AvailableClusterResource

Short names: available

Scope: Namespaced
Version: v1alpha1

Contains information about cluster-wide resources available to the project and the value used by default.

The resource is automatically created and updated by the `multitenancy-manager` controller in the project namespace for each registered GrantableClusterResourceDefinition. It is deleted from the project namespaces when the corresponding GrantableClusterResourceDefinition is deleted. The AvailableClusterResource name matches the name of the corresponding GrantableClusterResourceDefinition.

Information about available cluster-wide resources and the default value is stored in the `status` field.

For more information about viewing available cluster-wide resources, refer to Usage examples.

> **Warning.** The resource is read-only. Manual creation of AvailableClusterResource is prohibited.

- **status**
  object

  Cluster-wide resource availability status for the project.

  - **status.available**
    array of objects

    List of cluster-wide resources available to the project, indicating the resource used by default. Each resource is listed once.

    Information about a cluster-wide resource available to the project.

    - **status.available.default**
      boolean

      Indicates that the cluster-wide resource is used by default in the project.

    - **status.available.fields**
      object

      Values of the cluster-wide resource fields listed in `spec.catalogFields` of the corresponding GrantableClusterResourceDefinition, keyed by field name.

      A value keeps its JSON type: string, number, boolean, object or array. A field whose value is missing in the resource or is longer than 1024 bytes is not shown. If `status.available` with the fields of all resources would take more than 512 KiB in JSON serialization, `fields` is absent in every entry; the names and the default are still listed.

    - **status.available.name**
      string

      Required value

      Name of the cluster-wide resource available to the project.

  - **status.availableCount**
    integer

    Number of cluster-wide resources available to the project.

    Used when displaying the resource list. Names of available cluster-wide resources are listed in `status.available`.

  - **status.default**
    string

    Name of the cluster-wide resource used by default in the project.

    The value can be automatically assigned to the corresponding field of a resource being created if the field is empty or the specified value is unavailable to the project. The field remains empty if no default value is configured or the default value is unavailable to the project.

  - **status.grantedResourceKind**
    string

    Kind of the cluster-wide resource.

  - **status.observedGeneration**
    integer

    Resource generation based on which the controller generated the current `status`.

ClusterResourceGrantPolicy

Short names: crgp

Scope: Cluster

- v1alpha2
- v1alpha1

Defines a policy for project access to cluster-wide resources and the values used by default.

The resource applies to the project namespaces selected by `spec.projectSelector` (by the labels of the Project) and `spec.namespaceSelector` (by the labels of the namespace). For each type of cluster-wide resource, the policy allows you to configure allowed and denied resources, as well as their availability rules

and default values.

The cluster-wide resource type is specified using `spec.resources[].resourceName`, which references the corresponding GrantableClusterResourceDefinition.

The policy is created by a cluster administrator.

For configuration examples, refer to Usage examples.

- **spec**
  object

  Required value

  ClusterResourceGrantPolicy parameters.

  - **spec.namespaceSelector**
    object

    Label selector for project namespaces to which the policy applies.

    The selector is matched against the labels of the namespace. When `spec.projectSelector` is also specified, both must match. If the selector is not specified, all namespaces of the selected projects are covered. An empty selector matches all namespaces. The namespaces of the virtual projects `deckhouse` and `default` are never covered.

    The value of the `projects.deckhouse.io/project-namespace` label is the name that a project user gives to a ProjectNamespace. Without `spec.projectSelector`, a selector on this label also covers the namespace of a ProjectNamespace with the same name in any other project.

    An additional namespace of a project is created by a project user. It carries the labels set by the module and the labels of the project template, which it inherits from the main namespace; a label set by hand on the main namespace is not inherited. A policy that restricts resources by such a label does not cover a new additional namespace.

    - **spec.namespaceSelector.matchExpressions**
      array of objects

      List of label selector expressions.

      All expressions must match.

      Label selector expression.

      - **spec.namespaceSelector.matchExpressions.key**
        string

        Required value

        Label key to which the expression applies.

      - **spec.namespaceSelector.matchExpressions.operator**
        string

        Required value

        Operator that defines the relationship between the key and the set of values.

      - **spec.namespaceSelector.matchExpressions.values**
        array of strings

        List of label values.

        For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

    - **spec.namespaceSelector.matchLabels**
      object

      Set of namespace labels to which the policy applies.

      All specified labels must match.

  - **spec.projectSelector**
    object

    Label selector for projects to which the policy applies.

    The selector is matched against the labels of the Project object. The policy applies to every namespace of a selected project, including an additional namespace created later, unless `spec.namespaceSelector` narrows the choice. If the selector is not specified, projects are not chosen by their labels. An empty selector matches all projects, except the virtual projects

`deckhouse` and `default`, whose namespaces are never covered.

If neither `spec.projectSelector` nor `spec.namespaceSelector` is specified, the policy does not apply to any projects on its own. Such a policy is a library policy that a ProjectTemplate applies through `spec.grantPolicies`.

- **spec.projectSelector.matchExpressions**
  array of objects

  List of label selector expressions.

  All expressions must match.

  Label selector expression.

  - **spec.projectSelector.matchExpressions.key**
    string

    Required value

    Label key to which the expression applies.

  - **spec.projectSelector.matchExpressions.operator**
    string

    Required value

    Operator that defines the relationship between the key and the set of values.

  - **spec.projectSelector.matchExpressions.values**
    array of strings

    List of label values.

    For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.projectSelector.matchLabels**
  object

  Set of labels for projects to which the policy applies.

  All specified labels must match.

- **spec.resources**
  array of objects

  Required value

  Access settings for cluster-wide resources for the selected projects.

  Access settings for one type of cluster-wide resource.

  - **spec.resources.allowed**
    array of strings

    List of names of cluster-wide resources available to the projects.

    Combined with resources selected using `allowedSelector`. An empty list is not accepted: omit the field instead, or set `availabilityDefault: None` to make nothing available.

  - **spec.resources.allowedSelector**
    object

    Label selector for cluster-wide resources available to the projects.

    Combined with resources specified in `allowed`. Applies only to cluster-wide resources represented by Kubernetes objects. An empty selector is not accepted: omit the field instead, or set `availabilityDefault` to open or lock the resource without a list.

    - **spec.resources.allowedSelector.matchExpressions**
      array of objects

      List of label selector expressions.

      All expressions must match.

      Label selector expression.

      - **spec.resources.allowedSelector.matchExpressions.key**
        string

        Required value

Label key to which the expression applies.

- **spec.resources.allowedSelector.matchExpressions.operator**
  string

  Required value

  Operator that defines the relationship between the key and the set of values.

- **spec.resources.allowedSelector.matchExpressions.values**
  array of strings

  List of label values.

  For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.resources.allowedSelector.matchLabels**
  object

  Set of labels for cluster-wide resources.

  All specified labels must match.

- **spec.resources.availabilityDefault**
  string

  Determines the availability of cluster-wide resources that are not explicitly specified using `allowed`, `allowedSelector`, `denied`, or `deniedSelector`.

  Overrides `defaultAvailability` of the corresponding GrantableClusterResourceDefinition for the projects selected by the policy.

  `All` makes all such cluster-wide resources available, while `None` makes them unavailable.

  Usually, you do not need to specify this parameter. If `allowed` or `allowedSelector` is set and `availabilityDefault` is not specified, access to all other cluster-wide resources of this type is automatically denied. Set this parameter explicitly only to change availability without specifying a list of allowed resources: `All` to make all resources of the type available, or `None` to make them unavailable.

  Allowed values: `All`, `None`

- **spec.resources.default**
  string

  Name of the cluster-wide resource used by default in the selected projects.

  Overrides the value determined using `defaultFrom` in the corresponding GrantableClusterResourceDefinition.

- **spec.resources.denied**
  array of strings

  List of names of cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`.

- **spec.resources.deniedSelector**
  object

  Label selector for cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`. Applies only to cluster-wide resources represented by Kubernetes objects. An empty selector is not accepted.

  - **spec.resources.deniedSelector.matchExpressions**
    array of objects

    List of label selector expressions.

    All expressions must match.

    Label selector expression.

    - **spec.resources.deniedSelector.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.resources.deniedSelector.matchExpressions.operator**

string

Required value

Operator that defines the relationship between the key and the set of values.

- **spec.resources.deniedSelector.matchExpressions.values**
  array of strings

  List of label values.

  For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.resources.deniedSelector.matchLabels**
  object

  Set of labels for cluster-wide resources.

  All specified labels must match.

- **spec.resources.resourceName**
  string

  Required value

  Name of the GrantableClusterResourceDefinition for which the access settings are configured.

- **status**
  object

  ClusterResourceGrantPolicy status.

  - **status.conditions**
    array of objects

    Current state of the ClusterResourceGrantPolicy.

    Information about one aspect of the current resource state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Required value

      Message with additional information about the condition state.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Resource generation on which the condition state is based.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      Reason for the last change in the condition state.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

      Required value

      Condition state.

      Allowed values: `True`, `False`, `Unknown`

        - **status.conditions.type**
          string

          Required value

          Condition type.

          Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

          Maximum length: `316`

      - **status.observedGeneration**
        integer

        Latest resource generation processed by the controller.

Deprecated resource. Support for the resource might be removed in a later release.

Defines a policy for project access to cluster-wide resources and the values used by default.

The resource applies to the project namespaces whose labels match `spec.projectSelector`. A policy written through `multitenancy.deckhouse.io/v1alpha2` may also select projects by the labels of the Project; this version shows that selector as JSON in the `multitenancy.deckhouse.io/v1alpha2-project-selector` annotation. For each type of cluster-wide resource, the policy allows you to configure allowed and denied resources, as well as their availability rules and default values.

The cluster-wide resource type is specified using `spec.resources[].resourceName`, which references the corresponding GrantableClusterResourceDefinition.

The policy is created by a cluster administrator.

For configuration examples, refer to Usage examples.

- **spec**
  object

  Required value

  ClusterResourceGrantPolicy parameters.

  - **spec.projectSelector**
    object

    Label selector for project namespaces to which the policy applies.

    The selector is matched against the labels of the namespace, the same as `spec.namespaceSelector` of `multitenancy.deckhouse.io/v1alpha2`, which is the version to use.

    If the selector is not specified and the `multitenancy.deckhouse.io/v1alpha2-project-selector` annotation is absent, the policy does not apply to any projects. An empty selector matches all project namespaces, except the namespaces of the virtual projects `deckhouse` and `default`.

    The annotation holds `spec.projectSelector` of `v1alpha2`, which is matched against the labels of the Project. Keep the annotation when you update the policy through this version.

    - **spec.projectSelector.matchExpressions**
      array of objects

      List of label selector expressions.

      All expressions must match.

      Label selector expression.

      - **spec.projectSelector.matchExpressions.key**
        string

        Required value

        Label key to which the expression applies.

      - **spec.projectSelector.matchExpressions.operator**
        string

        Required value

        Operator that defines the relationship between the key and the set of values.

      - **spec.projectSelector.matchExpressions.values**
        array of strings

List of label values.

For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.projectSelector.matchLabels**
  object

  Set of namespace labels to which the policy applies.

  All specified labels must match.

- **spec.resources**
  array of objects

  Required value

  Access settings for cluster-wide resources for the selected projects.

  Access settings for one type of cluster-wide resource.

  - **spec.resources.allowed**
    array of strings

    List of names of cluster-wide resources available to the projects.

    Combined with resources selected using `allowedSelector`. An empty list is not accepted: omit the field instead, or set `availabilityDefault: None` to make nothing available.

  - **spec.resources.allowedSelector**
    object

    Label selector for cluster-wide resources available to the projects.

    Combined with resources specified in `allowed`. Applies only to cluster-wide resources represented by Kubernetes objects. An empty selector is not accepted: omit the field instead, or set `availabilityDefault` to open or lock the resource without a list.

    - **spec.resources.allowedSelector.matchExpressions**
      array of objects

      List of label selector expressions.

      All expressions must match.

      Label selector expression.

      - **spec.resources.allowedSelector.matchExpressions.key**
        string

        Required value

        Label key to which the expression applies.

      - **spec.resources.allowedSelector.matchExpressions.operator**
        string

        Required value

        Operator that defines the relationship between the key and the set of values.

      - **spec.resources.allowedSelector.matchExpressions.values**
        array of strings

        List of label values.

        For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

    - **spec.resources.allowedSelector.matchLabels**
      object

      Set of labels for cluster-wide resources.

      All specified labels must match.

  - **spec.resources.availabilityDefault**
    string

    Determines the availability of cluster-wide resources that are not explicitly specified using `allowed`, `allowedSelector`, `denied`, or `deniedSelector`.

    Overrides `defaultAvailability` of the corresponding GrantableClusterResourceDefinition

for the projects selected by the policy.

`All` makes all such cluster-wide resources available, while `None` makes them unavailable.

Usually, you do not need to specify this parameter. If `allowed` or `allowedSelector` is set and `availabilityDefault` is not specified, access to all other cluster-wide resources of this type is automatically denied. Set this parameter explicitly only to change availability without specifying a list of allowed resources: `All` to make all resources of the type available, or `None` to make them unavailable.

Allowed values: `All`, `None`

- **spec.resources.default**
  string

  Name of the cluster-wide resource used by default in the selected projects.

  Overrides the value determined using `defaultFrom` in the corresponding GrantableClusterResourceDefinition.

- **spec.resources.denied**
  array of strings

  List of names of cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`.

- **spec.resources.deniedSelector**
  object

  Label selector for cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`. Applies only to cluster-wide resources represented by Kubernetes objects. An empty selector is not accepted.

  - **spec.resources.deniedSelector.matchExpressions**
    array of objects

    List of label selector expressions.

    All expressions must match.

    Label selector expression.

    - **spec.resources.deniedSelector.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.resources.deniedSelector.matchExpressions.operator**
      string

      Required value

      Operator that defines the relationship between the key and the set of values.

    - **spec.resources.deniedSelector.matchExpressions.values**
      array of strings

      List of label values.

      For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

  - **spec.resources.deniedSelector.matchLabels**
    object

    Set of labels for cluster-wide resources.

    All specified labels must match.

- **spec.resources.resourceName**
  string

  Required value

  Name of the GrantableClusterResourceDefinition for which the access settings are configured.

- **status**
  object

ClusterResourceGrantPolicy status.

- **status.conditions**
  array of objects

  Current state of the ClusterResourceGrantPolicy.

  Information about one aspect of the current resource state.

  - **status.conditions.lastTransitionTime**
    string

    Required value

    Time when the condition last changed.

  - **status.conditions.message**
    string

    Required value

    Message with additional information about the condition state.

    Maximum length: `32768`

  - **status.conditions.observedGeneration**
    integer

    Resource generation on which the condition state is based.

    Allowed values: `0 <= X`

  - **status.conditions.reason**
    string

    Required value

    Reason for the last change in the condition state.

    Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

    Length: `1..1024`

  - **status.conditions.status**
    string

    Required value

    Condition state.

    Allowed values: `True`, `False`, `Unknown`

  - **status.conditions.type**
    string

    Required value

    Condition type.

    Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

    Maximum length: `316`

- **status.observedGeneration**
  integer

  Latest resource generation processed by the controller.

GrantableClusterResourceDefinition

Short names: gcrd

Scope: Cluster
Version: v1alpha1

Registers a type of cluster-wide resource for managing project access to it.

GrantableClusterResourceDefinition defines a cluster-wide resource for which access can be configured using ClusterResourceGrantPolicy, as well as the baseline rules for its availability and use of a default value.

The resource can register both Kubernetes objects, such as StorageClass or ClusterRole, and values that are not represented by separate Kubernetes objects.

Places where the cluster-wide resource is used in other Kubernetes resources are registered separately using GrantableClusterResourceReference.

GrantableClusterResourceDefinition is created by DP or a module developer. Custom definitions can be created to manage access to additional types of cluster-wide resources.

For more information, refer to “Usage examples”.

- **spec**
  object

  Required value

  GrantableClusterResourceDefinition parameters.

  - **spec.catalogFields**
    array of objects

    Fields of the cluster-wide resources shown to projects in AvailableClusterResource.

    For each available resource, the controller reads the value of every listed field from the resource and puts it into `status.available[].fields` of AvailableClusterResource under the specified name.

    Rules:

    - Applies only to definitions with `grantedResource`, which must be cluster-scoped. For values that are not represented by Kubernetes objects, the parameter is not allowed.
    - At most 10 fields can be listed.
    - A field whose value is missing in the resource is not shown.
    - A value longer than 1024 bytes in JSON serialization is not shown.
    - If `status.available` of one AvailableClusterResource with the fields of all resources would take more than 512 KiB in JSON serialization, no fields are shown in it.

    The paths are checked when the object is created or updated, and an invalid entry is refused. The result for the stored object is shown in the `CatalogFieldsValid` condition. Values left out for their size are reported by `CatalogFieldsSkipped` warning events on the GrantableClusterResourceDefinition.

    > **i** **Warning.** The values are visible to all users of every project the resource is available to. List only non-secret data.

    Field of the cluster-wide resource shown to projects.

    - **spec.catalogFields.name**
      string

      Required value

      Name of the field in `status.available[].fields` of AvailableClusterResource, in lowerCamelCase (for example, `reclaimPolicy`).

      Pattern: `^[a-z][a-zA-Z0-9]*$`

      Maximum length: `63`

    - **spec.catalogFields.path**
      string

      Required value

      Path to the value in the cluster-wide resource, for example `$.provisioner`.

      Must be an RFC 9535 JSONPath singular query of at most 256 characters that selects exactly one value: member names and array indexes only, without wildcards (`*`), descendant segments (`..`), slices or filters.

      Paths that read `metadata.managedFields` or the `kubectl.kubernetes.io/last-applied-configuration` annotation, or a parent that contains them (for example, `$.metadata`), are not allowed: these fields hold a copy of the whole resource.

      Length: `1..256`

  - **spec.defaultAvailability**
    string

    Baseline availability of cluster-wide resources to projects.

    Allowed values:

- `All`: Resources are available by default.
- `None`: Resources are unavailable by default.

The value applies if availability is not determined by ClusterResourceGrantPolicy.

Default: `All`

Allowed values: `All`, `None`

- **spec.defaultFrom**
  object

  Settings for selecting an object of a cluster-wide resource to use as the default.

  An object is marked as the default when one of the specified annotations is set to exactly `true`. The selected object's name is used as the default value if one is not set using ClusterResourceGrantPolicy. Other values, such as `false` or `True`, do not mark the object. If several objects are marked, the most recently created one is selected; if they have the same creation time, the first by name is selected. This is how Kubernetes picks the default StorageClass. The `DefaultFromResolved` condition in the status shows the selection result.

  - **spec.defaultFrom.additionalAnnotationKeys**
    array of strings

    Keys of additional annotations that, like `annotationKey`, mark an object of the cluster-wide resource as the default, for example, the older `storageclass.beta.kubernetes.io/is-default-class`.

  - **spec.defaultFrom.annotationKey**
    string

    Key of the annotation that marks the cluster-wide resource used by default; only the value `true` counts.

- **spec.enforcement**
  string

  Method for checking cluster-wide resource availability.

  Allowed values:

  - `Managed`: The check is performed by `multitenancy-manager` webhooks.
  - `External`: The check is performed by an external mechanism, such as a module webhook.

  Default: `Managed`

  Allowed values: `Managed`, `External`

- **spec.excluded**
  array of objects

  Filters for excluded cluster-wide resources that are always unavailable to projects regardless of ClusterResourceGrantPolicy settings.

  A resource is considered excluded if it matches at least one of the specified filters.

  Filter for cluster-wide resources by name and labels.

  - **spec.excluded.matchExpressions**
    array of objects

    List of label selector expressions for cluster-wide resources.

    All expressions must match.

    Label selector expression.

    - **spec.excluded.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.excluded.matchExpressions.operator**
      string

      Required value

      Operator that defines the relationship between the key and the set of values.

    - **spec.excluded.matchExpressions.values**

array of strings

List of label values.

For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.excluded.matchLabels**
  object

  Set of labels for cluster-wide resources.

  All specified labels must match.

- **spec.excluded.names**
  array of strings

  List of cluster-wide resource names.

- **spec.grantedResource**
  object

  Kubernetes cluster-wide resource for which access is configured.

  Must be a cluster-scoped resource: a namespaced one (for example, Secret) is refused when the object is created or updated, and a definition stored with one is not processed, which the `GrantedResourceValid` condition shows. A resource that is not served yet (its CRD is not installed) is accepted and checked once it is served.

  If the parameter is not specified, access management applies to values that are not represented by separate Kubernetes objects.

  - **spec.grantedResource.apiGroup**
    string

    API group of the cluster-wide resource.

    For the core API group, specify an empty string (`""`).

  - **spec.grantedResource.kind**
    string

    Kind of the cluster-wide resource (for example, StorageClass).

- **spec.reserved**
  array of objects

  Filters for reserved cluster-wide resources that are unavailable to projects by default.

  A reserved resource becomes available to a project only if a ClusterResourceGrantPolicy that applies to the project lists it in `allowed` or matches it with `allowedSelector`. `availabilityDefault: All` in a policy and `defaultAvailability` do not make it available. The `excluded` filters and the `denied` and `deniedSelector` values of a policy take precedence over `allowed` and `allowedSelector`.

  A resource is considered reserved if it matches at least one of the specified filters.

  Filter for cluster-wide resources by name and labels.

  - **spec.reserved.matchExpressions**
    array of objects

    List of label selector expressions for cluster-wide resources.

    All expressions must match.

    Label selector expression.

    - **spec.reserved.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.reserved.matchExpressions.operator**
      string

      Required value

      Operator that defines the relationship between the key and the set of values.

    - **spec.reserved.matchExpressions.values**

array of strings

List of label values.

For the `In` and `NotIn` operators, the list must contain at least one value. For the `Exists` and `DoesNotExist` operators, the list must be empty.

- **spec.reserved.matchLabels**
  object

  Set of labels for cluster-wide resources.

  All specified labels must match.

- **spec.reserved.names**
  array of strings

  List of cluster-wide resource names.

- **status**
  object

  GrantableClusterResourceDefinition status.

  - **status.conditions**
    array of objects

    Current state of the GrantableClusterResourceDefinition.

    - `GrantedResourceValid`: whether `spec.grantedResource` can be processed. `True` with the reason `ClusterScoped` for a cluster-scoped resource or `ValueBacked` for a definition without `grantedResource`. `False` with the reason `Namespaced` for a namespaced resource: the definition is not processed, no project gets its AvailableClusterResource, and the grant webhooks skip the references to it: their values are not checked. `Unknown` with the reason `KindNotServed` while the resource is not served (for example, its CRD is not installed yet), or `MappingFailed` if its scope could not be determined.
    - `CatalogFieldsValid`: whether `spec.catalogFields` pass the checks applied when the object is created or updated: the paths are valid singular queries that do not read `metadata.managedFields` or the `kubectl.kubernetes.io/last-applied-configuration` annotation, and the definition has `grantedResource` (reasons `Valid`, `InvalidCatalogFields`; the message lists all problems). A definition without `catalogFields` has the condition `True`. `False` means the object was stored bypassing these checks; the invalid entries are not shown in AvailableClusterResource.
    - `DefaultFromResolved`: result of selecting an object of the specified cluster-wide resource with `spec.defaultFrom`. `True` with reason `Found` means that one object is marked. The message names this object. `True` with reason `SeveralDefaults` means that several objects are marked. The message lists the marked objects and names the selected object. `False` with reason `NoDefault` means that no object has any of the specified annotations set to `true`. A project then gets a default only from a ClusterResourceGrantPolicy. `Unknown` with reason `ListFailed` means that the objects could not be read. When an object is selected, a project uses its name as the default. This happens when no ClusterResourceGrantPolicy sets a default for this cluster-wide resource and the object is available to the project. Annotation changes appear here within a few minutes. A definition without `defaultFrom` has no condition.

    Information about one aspect of the current resource state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Required value

      Message with additional information about the condition state.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Resource generation on which the condition state is based.

      Allowed values: `0 <= X`

    - **status.conditions.reason**

string

Required value

Reason for the last change in the condition state.

Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Condition state.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.observedGeneration**
  integer

  Latest resource generation processed by the controller.

- **status.referenceCount**
  integer

  Number of GrantableClusterResourceReference resources associated with this definition.

- **status.references**
  array of objects

  List of GrantableClusterResourceReference resources associated with this definition.

  Information about a GrantableClusterResourceReference associated with this definition.

  - **status.references.name**
    string

    Required value

    Name of the GrantableClusterResourceReference.

  - **status.references.resources**
    array of strings

    Types of Kubernetes resources that contain registered references to the cluster-wide resource.

GrantableClusterResourceReference

Short names: gcrr

Scope: Cluster
Version: v1alpha1

Registers where a cluster-wide resource is used in other Kubernetes resources.

GrantableClusterResourceReference defines Kubernetes resources and fields that contain references to a cluster-wide resource registered using GrantableClusterResourceDefinition. For such fields, you can configure availability checks for the specified cluster-wide resource and automatic default value assignment.

One GrantableClusterResourceDefinition can be associated with multiple GrantableClusterResourceReference resources if the cluster-wide resource is used in different resources or fields.

GrantableClusterResourceReference is created by a module developer when adding a reference to a registered cluster-wide resource.

For more information about registering references to cluster-wide resources, refer to “Usage examples”.

- **spec**
  object

  Required value

  GrantableClusterResourceReference parameters.

  - **spec.fieldPaths**
    array of objects

    Required value

    List of Kubernetes resource fields that contain references to the cluster-wide resource.

    For each field, you can limit applicability to specific resources, API groups and versions, define a matching condition, and configure default value assignment.

    At least one item is required. The `resources`, `apiGroups` and `apiVersions` of an item must be a subset of the corresponding lists in `spec.rule` (`*` in `spec.rule` accepts any value). Every combination of API group, version and resource that `spec.rule` matches must be covered by at least one item, otherwise requests for it would not be checked at all. Either give items scoped to every such combination, or add an item without `resources`, `apiGroups` and `apiVersions` as the fallback for all other cases. An object that violates either rule is rejected.

    Settings for a field that contains a reference to the cluster-wide resource.

    If a request matches multiple items, the most specific one applies: `resources` takes precedence over `apiGroups`, and `apiGroups` takes precedence over `apiVersions`. `*` in `resources`, `apiGroups`, or `apiVersions` does not restrict the item and does not make it more specific. Items whose `match` condition does not hold for the object are skipped. Of equally restricted items, an item with `match` takes precedence over an item without it. If several items are equally specific, the first one in the list applies. An item without restrictions and without `match` is used in all other cases.

    Several items with the same `path` can therefore handle different objects of one resource. For example, an item with `match` on `$.spec['dataSource','dataSourceRef'].kind` and `defaulting: None` handles the PersistentVolumeClaims that restore or clone a volume, and an item without `match` and with `defaulting: Coerce` handles all other PersistentVolumeClaims.

    - **spec.fieldPaths.apiGroups**
      array of strings

      API groups to which the field settings apply.

      If the parameter is not specified, the settings apply to all API groups that match `spec.rule`.

    - **spec.fieldPaths.apiVersions**
      array of strings

      API versions to which the field settings apply.

      If the parameter is not specified, the settings apply to all API versions that match `spec.rule`.

    - **spec.fieldPaths.defaulting**
      string

      Default value assignment mode.

      Allowed values:

      - `None`: No value is assigned; only the availability of the specified cluster-wide resource is checked.
      - `FillEmpty`: The default value is assigned when an object is created if the field is empty.
      - `Coerce`: The default value is assigned when an object is created if the field is not set, if the author of the object did not set it (for example, the built-in Kubernetes admission controller filled in the default StorageClass of the cluster), or if the specified cluster-wide resource is unavailable to the project. Whether the author set the field is taken from `metadata.managedFields` of the request; an object without them counts as set by its author. A field explicitly set to an empty string is kept.

      `FillEmpty` and `Coerce` require `path` to point to a single field, so only a simple member path is accepted, for example `$.spec.storageClassName` or `$.metadata.annotations['cert-manager.io/cluster-issuer']`. A path with a wildcard, an array index or a filter is rejected with these modes and can only be used with `None`.

      Default: `None`

      Allowed values: `None`, `FillEmpty`, `Coerce`

- **spec.fieldPaths.match**
  object

  Condition for applying the field settings.

  If the condition does not match, the item is skipped: the next most specific item applies, and if there is none, availability checks and default value assignment are not performed for this field.

  Without `equals` and `in`, the condition matches when `fieldPath` points to a string value.

  - **spec.fieldPaths.match.equals**
    string

    Value that the field specified in `fieldPath` must match.

  - **spec.fieldPaths.match.fieldPath**
    string

    Required value

    JSONPath to the field whose value is checked by the condition.

    Must be a valid RFC 9535 JSONPath of at most 256 characters; an object with a path that does not parse is rejected.

    Maximum length: `256`

  - **spec.fieldPaths.match.in**
    array of strings

    List of allowed values for the field specified in `fieldPath`.

- **spec.fieldPaths.message**
  string

  Text appended to the rejection of an object whose field references a cluster-wide resource that is unavailable to the project, and to the rejection of an object that leaves the reference out in a place `missingPath` selects.

  Use it to tell the user what to do, for example why the value cannot be replaced with the project default.

  Maximum length: `512`

- **spec.fieldPaths.missingPath**
  string

  JSONPath to the places of the object where a reference to the cluster-wide resource is expected but not given.

  Use it for a list of templates whose elements can leave the field from `path` unset, for example `$.spec.volumeClaimTemplates[?!@.spec.storageClassName]` selects the StatefulSet volume claim templates without `storageClassName`. Such an element gets the cluster-wide default, for example the default StorageClass of the cluster, which `path` does not see.

  If the GrantableClusterResourceDefinition has `defaultFrom`, the cluster-wide default it finds is checked as if the object referenced it: an object in which the path selects anything is rejected if that default is unavailable to the project, and admitted if there is no default. Without `defaultFrom`, such an object is rejected if a ClusterResourceGrantPolicy with an entry for the cluster-wide resource applies to the namespace, and admitted otherwise. On update, only the places the old object did not have are taken into account.

  Must be a valid RFC 9535 JSONPath of at most 256 characters; an object with a path that does not parse is rejected.

  Maximum length: `256`

- **spec.fieldPaths.path**
  string

  Required value

  JSONPath to the string field containing the name of the cluster-wide resource.

  The path can also point to an annotation, for example, `$.metadata.annotations['cert-manager.io/cluster-issuer']`.

  Must be a valid RFC 9535 JSONPath of at most 256 characters; an object with a path that does not parse is rejected, whatever `defaulting` is set to. A field name that is not a plain identifier (contains `-`, `.` or `/`, or starts with a digit) must be written in brackets, for

example `$.spec['foo-bar']`.

An availability check evaluates the full JSONPath, including wildcards, array indexes and filters. Default value assignment does not: with `defaulting: FillEmpty` or `defaulting: Coerce`, only a simple member path is accepted.

Maximum length: `256`

- **spec.fieldPaths.resources**
  array of strings

  Kubernetes resource names in plural, for example, `pods` or `cronjobs`, to which the field settings apply.

  If the parameter is not specified, the settings apply to all resources that match `spec.rule`. `*` has the same effect as omitting the parameter: it does not restrict the item and does not make it more specific.

  - **Element of the array**
    string

- **spec.grantableClusterResourceName**
  string

  Required value

  Name of the GrantableClusterResourceDefinition that registers the cluster-wide resource referenced by the fields.

- **spec.rule**
  object

  Required value

  Rule for selecting Kubernetes resources to which GrantableClusterResourceReference applies.

  - **spec.rule.apiGroups**
    array of strings

    Required value

    API groups of Kubernetes resources. The core group is the empty string.

    `*` is not accepted: the admission webhook is registered for the listed groups only, so a wildcard would silently intercept nothing.

  - **spec.rule.apiVersions**
    array of strings

    Required value

    API versions of Kubernetes resources.

    Use `*` to select all API versions.

  - **spec.rule.resources**
    array of strings

    Required value

    Kubernetes resource names in plural form (for example, `ingresses`).

- **status**
  object

  GrantableClusterResourceReference status.

  - **status.bound**
    boolean

    Indicates that the GrantableClusterResourceDefinition specified in `spec.grantableClusterResourceName` exists.

  - **status.conditions**
    array of objects

    Current state of GrantableClusterResourceReference.

    - `Bound`: whether the GrantableClusterResourceDefinition specified in `spec.grantableClusterResourceName` exists (reasons `Resolved`, `UnknownResource`).
    - `FieldPathsValid`: whether `spec.fieldPaths` pass the checks applied when the object is created or updated: the paths are valid, defaulting paths are simple member paths, item scopes are within `spec.rule`, and every combination `spec.rule` matches is covered

(reasons `Valid`, `InvalidFieldPaths`; the message lists all problems). `False` means the object was stored bypassing these checks; the invalid paths are skipped during availability checks.

Information about one aspect of the current resource state.

- **status.conditions.lastTransitionTime**
  string

  Required value

  Time when the condition last changed.

- **status.conditions.message**
  string

  Required value

  Message with additional information about the condition state.

  Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  Resource generation on which the condition state is based.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  Reason for the last change in the condition state.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Condition state.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.observedGeneration**
  integer

  Latest resource generation processed by the controller.

ProjectNamespace

Short names: pns

Scope: Namespaced
Version: v1alpha3

Describes an additional namespace of a project.

Available in the commercial editions of Deckhouse Platform. In Deckhouse Platform Open, a ProjectNamespace cannot be created, and the controller neither creates nor manages its namespace.

It must be created in the project's main namespace (the namespace whose name equals the project name). The controller creates a namespace named `<project>-<spec.name>`, labels it with the project ownership labels, and adds it to the parent `Project.status.namespaces` with kind `Additional`. Deleting the ProjectNamespace removes that namespace.

- **spec**
  object

  Required value

  - **spec.name**
    string

    Required value

    Suffix of the resulting namespace. The created namespace is `<project>-<name>`, where `<project>` is the main namespace this object lives in.

    The resulting name `<project>-<name>` must be no longer than 63 characters.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Length: `1..61`

- **status**
  object
  - **status.conditions**
    array of objects

    Current state of the object.

    - **status.conditions.lastProbeTime**
      string

      Time of the last check of the condition.

    - **status.conditions.lastTransitionTime**
      string

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Message with details of the condition state.

    - **status.conditions.status**
      string

      Condition state: `True`, `False` or `Unknown`.

    - **status.conditions.type**
      string

      Condition type.

  - **status.namespace**
    string

    The namespace the controller created for this ProjectNamespace.

  - **status.observedGeneration**
    integer

    Last generation of the object processed by the controller.

ProjectQuotaStatus

Short names: pqs

Scope: Cluster
Version: v1alpha1

Limits and usage of the quota of a project, summed over all namespaces of the project.

The object is read-only. The controller creates it from `Project.spec.quota`, keeps it up to date and deletes it when the field is cleared or the project is deleted; set the quota of a project in `Project.spec.quota`. The name of the object is the name of the project.

The controller puts the `d8-project-quota` ResourceQuota with the limits of `spec.hard` into every namespace of the project, the ones created later included, so that the usage of each namespace is counted by its own ResourceQuota and no namespace alone uses more than the whole project may. The status shows the usage summed over the namespaces.

- **spec**
  object

Required value

- **spec.hard**
  object

  Limits of the project, copied by the controller from `Project.spec.quota`, in the format of `ResourceQuota.spec.hard`, for example `requests.cpu: "100"`.

  Accepted keys are the ones a ResourceQuota counts in a namespace: `cpu`, `memory`, `requests.*`, `limits.*`, `pods`, `services`, `services.loadbalancers`, `services.nodeports`, `persistentvolumeclaims`, `configmaps`, `secrets`, `replicationcontrollers`, `ephemeral-storage`, the storage class and device class keys, the requests of extended resources and `count/<RESOURCE>[.<GROUP>]` for a namespaced resource the API server serves. `projectNamespaces` is an alias of `count/projectnamespaces.deckhouse.io`, the number of ProjectNamespace objects of the project.

  Where the `virtualization` module is enabled, the virtual machines of the project are limited by their specification with `virtualmachines.virtualization.deckhouse.io/cores`, `virtualmachines.virtualization.deckhouse.io/cpu`, `virtualmachines.virtualization.deckhouse.io/memory`, `virtualmachines.virtualization.deckhouse.io/gpus` and `<GPU_CLASS>.gpuclass.virtualization.deckhouse.io/gpus`; no ResourceQuota counts these keys, so they do not go into `d8-project-quota`. The Services of type LoadBalancer are limited by class, pool and address with `<CLASS>.loadbalancerclass.network.deckhouse.io/services`, `<CLASS>.loadbalancerclass.network.deckhouse.io/addresses`, `<POOL>.loadbalancerpool.network.deckhouse.io/addresses` and `loadbalancers.network.deckhouse.io/addresses`, which do not go into it either.

  `resourcequotas`, `count/resourcequotas`, the count of a cluster-scoped resource and a misspelled resource are refused. A `count/` key of a group the API server does not serve yet is accepted with a warning and counts nothing until the module that provides the group is enabled.

- **status**
  object
  - **status.conditions**
    array of objects

    State of the quota:

    - If `Enforced` is `True`, the quota applies in every namespace of the project. If `Enforced` is `False`, `reason` says why: `ControlPlaneNotManaged` (Deckhouse Platform does not manage the control plane, so the sum is not enforced), `Suspended` (the `quota-ledger-suspend` ConfigMap suspends the check of the sum), `WebhookMissing` (the webhook of the quota ledger is missing or its `caBundle` is not the CA of the ledger certificate), `WebhookMisconfigured` (that webhook is changed so that writes of `d8-project-quota` do not reach the ledger), `WebhookNotCalled` (some API server admits objects without calling that webhook), `KeyNotServed` (nothing counts a key: the API server does not serve the resource of a `count/` key yet, or the module that serves the resources of a key is disabled) or `KeyRollout` (the `d8-project-quota` ResourceQuota of some namespace is not there or not calculated with every limit yet).
    - If `Exceeded` is `True`, the usage of the project is above the quota on some key. The `reason` says how the usage got above the quota: `HardLowered` (the limit was set below the usage, by a change or when the quota was created) or `Overshoot` (the usage grew past the limit).
    - **status.conditions.lastTransitionTime**
      string

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Message with details of the condition state.

    - **status.conditions.observedGeneration**
      integer

      Generation of the object the condition was set for.

    - **status.conditions.reason**
      string

      Reason of the condition state.

    - **status.conditions.status**
      string

      Required value

      Condition state: `True`, `False` or `Unknown`.

        - **status.conditions.type**
          string

          Required value

          Condition type.

    - **status.hard**
      object

      Limits the controller applies, `spec.hard` with the alias expanded.

    - **status.namespaceCount**
      integer

      Number of namespaces of the project.

    - **status.observedGeneration**
      integer

      Last generation of the object the controller processed.

    - **status.topNamespaces**
      array of objects

      Namespaces closest to a limit, at most 50 of them. The usage of every namespace is in its `d8-project-quota` ResourceQuota.

        - **status.topNamespaces.name**
          string

          Required value

          Name of the namespace.

        - **status.topNamespaces.used**
          object

          Usage of the namespace.

    - **status.used**
      object

      Usage of the project, summed over all of its namespaces.

ProjectRoleBinding

Short names: prb

Scope: Namespaced
Version: v1alpha3

ProjectRoleBinding grants a role to subjects across all namespaces of a single project.

It must be created in the project’s main namespace (the namespace whose name equals the project name).

- **spec**
  object

  Required value

    - **spec.roleRef**
      object

      Required value

      Reference to the granted ClusterRole. Allowed prefixes: `d8:project:*`, `d8:namespace:*`, `d8:project-capability:*`, `d8:namespace-capability:*`, `d8:custom:*`.

        - **spec.roleRef.kind**
          string

          Required value

          Kind of the granted role.

          Allowed values: `ClusterRole`

        - **spec.roleRef.name**
          string

          Required value

Name of the granted ClusterRole.

- **spec.subjects**
  array of objects

  Required value

  Subjects (users, groups or service accounts) the role is granted to.

  - **spec.subjects.apiGroup**
    string

    API group of the subject: `rbac.authorization.k8s.io` for `User` and `Group`, empty for `ServiceAccount`.

  - **spec.subjects.kind**
    string

    Required value

    Kind of the subject.

    Allowed values: `User`, `Group`, `ServiceAccount`

  - **spec.subjects.name**
    string

    Required value

    Name of the subject: the user name (`User.spec.email`) for `User`, `Group.spec.name` for `Group`, the ServiceAccount name for `ServiceAccount`.

  - **spec.subjects.namespace**
    string

    Namespace of the ServiceAccount.

    If omitted, the binding names the ServiceAccount with this name in each namespace of the project.

- **status**
  object
  - **status.conditions**
    array of objects

    Current state of the object.

    - **status.conditions.lastProbeTime**
      string

      Time of the last check of the condition.

    - **status.conditions.lastTransitionTime**
      string

      Time when the condition last changed.

    - **status.conditions.message**
      string

      Message with details of the condition state.

    - **status.conditions.status**
      string

      Condition state: `True`, `False` or `Unknown`.

    - **status.conditions.type**
      string

      Condition type.

  - **status.observedGeneration**
    integer

    Last generation of the object processed by the controller.

Project

Scope: Cluster

- v1alpha3

- v1alpha2
- v1alpha1

Describes a project: an isolated environment for applications.

A project consists of the main namespace named after the project, the additional namespaces (ProjectNamespace), if any, and the settings, policies and objects that its ProjectTemplate describes. See Creating a project.

- **spec**
  object
  - **spec.administrators**
    array of objects

    Users and groups that get administrative access to the project.

    The controller grants them the `d8:project:admin` role through the `d8-administrators` ProjectRoleBinding. For `User`, the name is the user name (`User.spec.email`); for `Group`, it is `Group.spec.name`.

    Cannot be set together with `parameters.administrators`.

    - **spec.administrators.kind**
      string

      Required value

      Kind of the subject.

      Allowed values: `User`, `Group`

    - **spec.administrators.name**
      string

      Required value

      Subject name. For `User`, the user name (`User.spec.email`). For `Group`, `Group.spec.name` (the name in the token), not `metadata.name`. The name cannot contain control characters or line breaks and cannot start with `system:multitenancy-manager:`, the prefix of the users that the controller acts as. An update that keeps such a name as it was is accepted with a warning.

      Pattern: `^[^\x00-\x1f\x7f]+$`

  - **spec.description**
    string

    Arbitrary description of the project’s purpose. Specify an empty string if no description is required.

  - **spec.grants**
    object

    Cluster-wide resources available to the namespaces of the project, added to the ones its template grants.

    The controller turns them into managed ClusterResourceGrantPolicy objects named `project-<PROJECT_NAME>-*`, owned by the project. Changing the field needs the right to create ClusterResourceGrantPolicy objects. `deckhouse.io/v1alpha2` cannot describe the field, so a write of a project with grants through that version, a write of its status included, is refused.

    - **spec.grants.grantPolicies**
      array of strings

      References to library ClusterResourceGrantPolicy objects (created without `projectSelector` and `namespaceSelector`).

      The controller creates and maintains one managed policy per reference, `project-<PROJECT_NAME>-<POLICY_NAME>`, copying its resources and binding them to every namespace of the project.

    - **spec.grants.resources**
      array of objects

      Rules of access to cluster-wide resources for the namespaces of the project, in the format of `resources` of a ProjectTemplate.

      The controller turns them into the `project-<PROJECT_NAME>-inline` ClusterResourceGrantPolicy, which applies to every namespace of the project.

      Access settings for one type of cluster-wide resource.

      - **spec.grants.resources.allowed**

array of strings

List of names of cluster-wide resources available to the project.

Combined with the resources selected with `allowedSelector`.

- **spec.grants.resources.allowedSelector**
  object

  Label selector of cluster-wide resources available to the project.

  Combined with the resources listed in `allowed`. Applies only to cluster-wide resources represented by Kubernetes objects.

  - **spec.grants.resources.allowedSelector.matchExpressions**
    array of objects

    List of label selector expressions.

    All expressions must match.

    - **spec.grants.resources.allowedSelector.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.grants.resources.allowedSelector.matchExpressions.operator**
      string

      Required value

      Operator that defines the relationship between the key and the set of values.

    - **spec.grants.resources.allowedSelector.matchExpressions.values**
      array of strings

      List of label values.

  - **spec.grants.resources.allowedSelector.matchLabels**
    object

    Set of labels of cluster-wide resources.

    All specified labels must match.

- **spec.grants.resources.availabilityDefault**
  string

  Availability of the cluster-wide resources of this type that `allowed`, `allowedSelector`, `denied` and `deniedSelector` do not name.

  Overrides `defaultAvailability` of the GrantableClusterResourceDefinition. If `allowed` or `allowedSelector` is set and this field is not, all other resources of this type are unavailable to the project.

  Allowed values: `All`, `None`

- **spec.grants.resources.default**
  string

  Name of the cluster-wide resource used by default in the project.

  Overrides the value determined with `defaultFrom` in the GrantableClusterResourceDefinition.

- **spec.grants.resources.denied**
  array of strings

  List of names of cluster-wide resources unavailable to the project.

  Deny rules take precedence over `allowed` and `allowedSelector`.

- **spec.grants.resources.deniedSelector**
  object

  Label selector of cluster-wide resources unavailable to the project.

  Deny rules take precedence over `allowed` and `allowedSelector`.

  - **spec.grants.resources.deniedSelector.matchExpressions**
    array of objects

List of label selector expressions.

All expressions must match.

- **spec.grants.resources.deniedSelector.matchExpressions.key**
  string

  Required value

  Label key to which the expression applies.

- **spec.grants.resources.deniedSelector.matchExpressions.operator**
  string

  Required value

  Operator that defines the relationship between the key and the set of values.

- **spec.grants.resources.deniedSelector.matchExpressions.values**
  array of strings

  List of label values.

- **spec.grants.resources.deniedSelector.matchLabels**
  object

  Set of labels of cluster-wide resources.

  All specified labels must match.

- **spec.grants.resources.resourceName**
  string

  Required value

  Name of the GrantableClusterResourceDefinition that registers the type of cluster-wide resource.

- **spec.parameters**
  object

  Values of the parameters that the template declares in `parametersSchema`.

  The structured fields of the template take them through `{fromParam: <NAME>}`. `manifests` and `resourcesTemplate` get them in `{{ .parameters }}`, completed with the defaults of `parametersSchema`. The values are validated against `parametersSchema.openAPIV3Schema` of the template when the project is created or changed.

- **spec.projectTemplateName**
  string

  Name of the ProjectTemplate the project is created from.

  The `simple` template creates only the project namespace.

  Default: `simple`

- **spec.quota**
  object

  Resource quota of the whole project.

  The controller puts the `d8-project-quota` ResourceQuota into every namespace of the project, sums the usage over them and shows the limits and the usage in the read-only ProjectQuotaStatus object of the project. Clearing the field deletes that ProjectQuotaStatus.

  Changing the field needs the `update` verb on `projects/quota` in the `deckhouse.io` group for the project, which the `manage_quotas` capability of the `multitenancy-manager` module grants.

  Keys and values follow the `ResourceQuota.spec.hard` format, for example `requests.cpu: "100"`. Memory and storage quantities must include an explicit unit suffix (for example `requests.memory: 2Gi`); bare numbers are interpreted as bytes, so the admission webhook rejects a bare number other than 0. An update that leaves such a value as it was is accepted with a warning, and so is a create through `deckhouse.io/v1alpha2`, where such a value comes from `parameters.resourceQuota`.

  Cannot be set together with `parameters.resourceQuota`.

- **status**
  object
  - **status.conditions**
    array of objects

Current state of the project.

The condition types are described in Project status and diagnostics.

- **status.conditions.lastProbeTime**
  string

  Time of the last check of the condition.

- **status.conditions.lastTransitionTime**
  string

  Time when the condition last changed.

- **status.conditions.message**
  string

  Message with details of the condition state.

- **status.conditions.status**
  string

  Condition state: `True`, `False` or `Unknown`.

- **status.conditions.type**
  string

  Condition type.

- **status.handingOver**
  array of objects

  Objects of `manifests` that an upgrade of the project release is taking over.

  Each entry names the object and the release revision the upgrade is expected to deploy. The object stays in the cluster, and `manifestObjects` does not list it while the entry exists. The entry is removed once the upgrade deploys the object. If the upgrade does not deploy it, the object goes back to `manifests`. It gets the `projects.deckhouse.io/project` and `projects.deckhouse.io/source: manifests` labels back and loses the `heritage`, `app.kubernetes.io/managed-by` and `projects.deckhouse.io/project-template` labels, and `manifestObjects` lists it again.

  No object is handed over while the upgrade would also take an object that is not an object of `manifests` and exists without the `app.kubernetes.io/managed-by: Helm` label and the `meta.helm.sh/release-name` annotation of the release. The upgrade does not run then, and the `ResourcesUpgraded` condition names that object.

  - **status.handingOver.apiVersion**
    string

    API version of the object.

  - **status.handingOver.kind**
    string

    Kind of the object.

  - **status.handingOver.name**
    string

    Name of the object.

  - **status.handingOver.namespace**
    string

    Namespace of the object.

  - **status.handingOver.revision**
    integer

    Release revision the upgrade is expected to deploy.

- **status.keptObjects**
  array of objects

  Objects that the project release stopped holding and that were left in place instead of being deleted.

  These are the objects of a Helm text in `resourcesTemplate` after the project switched to structured fields, a quota kept for a handover after `.spec.quota` stopped covering it, and the objects of the release of the structured fields that `manifests` render. The release no longer manages them. They carry the `projects.deckhouse.io/kept-from-helm-template: "true"`

label, and while it is there, only a requester who may make the same change across the cluster may change or delete them.

An object leaves the list once it is deleted, the label is removed from it, the release renders it again, or `manifests` take it over. When the project is deleted, the cluster-scoped ones that still carry the label are deleted with it.

- **status.keptObjects.apiVersion**
  string

  API version of the object.

- **status.keptObjects.kind**
  string

  Kind of the object.

- **status.keptObjects.name**
  string

  Name of the object.

- **status.keptObjects.namespace**
  string

  Namespace of the object, empty for a cluster-scoped one.

- **status.manifestObjects**
  array of objects

  Objects that the controller applied from `manifests` of the project template, as the `system:multitenancy-manager:project:<PROJECT_NAME>` user. They carry the `projects.deckhouse.io/project` label with the project name and the `projects.deckhouse.io/source: manifests` label. An object that the template does not render any more is deleted and leaves the list. One that lost one of these labels by then, or whose namespace left the project, leaves the list and stays in place.

  - **status.manifestObjects.apiVersion**
    string

    API version of the object.

  - **status.manifestObjects.kind**
    string

    Kind of the object.

  - **status.manifestObjects.name**
    string

    Name of the object.

  - **status.manifestObjects.namespace**
    string

    Namespace of the object.

- **status.namespaces**
  array of objects

  Namespaces that belong to the project.

  - **status.namespaces.kind**
    string

    Kind of the namespace (Main or Additional).

  - **status.namespaces.name**
    string

    Name of the namespace.

- **status.observedGeneration**
  integer

  Last generation of the project processed by the controller.

- **status.resources**
  object

  Rendered and skipped resources.

    - **status.state**
      string

      State of the project: `Deployed` or `Error`. The conditions and their messages give the details.

    - **status.templateGeneration**
      integer

      Generation of the template the project was last rendered from.

    - **status.usage**
      object

      Current utilization of the project quota.

Describes a project: an isolated environment for applications.

A project consists of the main namespace named after the project, the additional namespaces (ProjectNamespace), if any, and the settings, policies and objects that its ProjectTemplate describes. See Creating a project.

- **spec**
  object
    - **spec.description**
      string

      Arbitrary description of the project’s purpose. Specify an empty string if no description is required.

    - **spec.parameters**
      object

      Values of the parameters that the template declares in `parametersSchema`.

      The structured fields of the template take them through `{fromParam: <NAME>}`. `manifests` and `resourcesTemplate` get them in `{{ .parameters }}`, completed with the defaults of `parametersSchema`. The values are validated against `parametersSchema.openAPIV3Schema` of the template when the project is created or changed.

    - **spec.projectTemplateName**
      string

      Name of the ProjectTemplate the project is created from.

      The `simple` template creates only the project namespace.

      Default: `simple`

- **status**
  object
    - **status.conditions**
      array of objects

      Current state of the project.

      The condition types are described in Project status and diagnostics.

        - **status.conditions.lastProbeTime**
          string

          Time of the last check of the condition.

        - **status.conditions.lastTransitionTime**
          string

          Time when the condition last changed.

        - **status.conditions.message**
          string

          Message with details of the condition state.

        - **status.conditions.status**
          string

          Condition state: `True`, `False` or `Unknown`.

        - **status.conditions.type**
          string

          Condition type.

    - **status.handingOver**

array of objects

Objects of `manifests` that an upgrade of the project release is taking over.

Each entry names the object and the release revision the upgrade is expected to deploy. The object stays in the cluster, and `manifestObjects` does not list it while the entry exists. The entry is removed once the upgrade deploys the object. If the upgrade does not deploy it, the object goes back to `manifests`. It gets the `projects.deckhouse.io/project` and `projects.deckhouse.io/source: manifests` labels back and loses the `heritage`, `app.kubernetes.io/managed-by` and `projects.deckhouse.io/project-template` labels, and `manifestObjects` lists it again.

No object is handed over while the upgrade would also take an object that is not an object of `manifests` and exists without the `app.kubernetes.io/managed-by: Helm` label and the `meta.helm.sh/release-name` annotation of the release. The upgrade does not run then, and the `ResourcesUpgraded` condition names that object.

- **status.handingOver.apiVersion**
  string

  API version of the object.

- **status.handingOver.kind**
  string

  Kind of the object.

- **status.handingOver.name**
  string

  Name of the object.

- **status.handingOver.namespace**
  string

  Namespace of the object.

- **status.handingOver.revision**
  integer

  Release revision the upgrade is expected to deploy.

- **status.keptObjects**
  array of objects

  Objects that the project release stopped holding and that were left in place instead of being deleted.

  These are the objects of a Helm text in `resourcesTemplate` after the project switched to structured fields, a quota kept for a handover after `.spec.quota` stopped covering it, and the objects of the release of the structured fields that `manifests` render. The release no longer manages them. They carry the `projects.deckhouse.io/kept-from-helm-template: "true"` label, and while it is there, only a requester who may make the same change across the cluster may change or delete them.

  An object leaves the list once it is deleted, the label is removed from it, the release renders it again, or `manifests` take it over. When the project is deleted, the cluster-scoped ones that still carry the label are deleted with it.

  - **status.keptObjects.apiVersion**
    string

    API version of the object.

  - **status.keptObjects.kind**
    string

    Kind of the object.

  - **status.keptObjects.name**
    string

    Name of the object.

  - **status.keptObjects.namespace**
    string

    Namespace of the object, empty for a cluster-scoped one.

- **status.manifestObjects**
  array of objects

Objects that the controller applied from `manifests` of the project template, as the `system:multitenancy-manager:project:<PROJECT_NAME>` user. They carry the `projects.deckhouse.io/project` label with the project name and the `projects.deckhouse.io/source: manifests` label. An object that the template does not render any more is deleted and leaves the list. One that lost one of these labels by then, or whose namespace left the project, leaves the list and stays in place.

- **status.manifestObjects.apiVersion**
  string

  API version of the object.

- **status.manifestObjects.kind**
  string

  Kind of the object.

- **status.manifestObjects.name**
  string

  Name of the object.

- **status.manifestObjects.namespace**
  string

  Namespace of the object.

- **status.namespaces**
  array of strings

  Used namespaces.

- **status.observedGeneration**
  integer

  Last generation of the project processed by the controller.

- **status.resources**
  object

  Rendered and skipped resources.

- **status.state**
  string

  State of the project: `Deployed` or `Error`. The conditions and their messages give the details.

- **status.templateGeneration**
  integer

  Generation of the template the project was last rendered from.

Deprecated resource. Support for the resource might be removed in a later release.

ProjectTemplate

Scope: Cluster

- v1alpha2
- v1alpha1

Describes the project template as a set of structured, declarative fields.

This resource is referenced by the Project custom resource. The `resources` and `grantPolicies` fields become ClusterResourceGrantPolicy objects that apply to the projects of this template. The other fields become objects in the namespaces of the projects.

Objects that no structured field describes are rendered from `manifests`. Instead of the fields that render objects, a template created through `deckhouse.io/v1alpha1` can carry a Helm template in `resourcesTemplate`. This form is deprecated, and no template can get a new `resourcesTemplate`.

The `virtual` name is reserved for the template of the virtual projects `default` and `deckhouse`: a ProjectTemplate with this name cannot be created.

- **spec**
  object
  - **spec.allowedGIDs**

    Range of group IDs allowed in the project containers. A literal `{min, max}` object or a parameter reference `{fromParam: "<name>"}`.

- **spec.allowedUIDs**

  Range of user IDs allowed in the project containers. A literal `{min, max}` object or a parameter reference `{fromParam: "<name>"}`.

- **spec.description**
  string

  Arbitrary description of the template’s purpose. Specify an empty string if no description is required.

- **spec.features**
  object

  Optional project capabilities.

  - **spec.features.monitoring**

    Enable extended monitoring for the project. A literal boolean or a parameter reference `{fromParam: "<name>"}`.

  - **spec.features.vulnerabilityScanning**

    Enable periodic vulnerability scans (Trivy) for the project. A literal boolean or a parameter reference `{fromParam: "<name>"}`.

- **spec.grantPolicies**
  array of strings

  References to library ClusterResourceGrantPolicy objects (created without `projectSelector` and `namespaceSelector`).

  The controller creates and maintains one managed policy per reference, copying its resources and binding them to the namespaces rendered from this template.

- **spec.logShipping**
  object

  Forwarding of the project logs to a cluster log destination.

  - **spec.logShipping.clusterDestinationRef**

    Name of the ClusterLogDestination to ship logs to. A literal string or a parameter reference `{fromParam: "<name>"}`.

- **spec.manifests**
  string

  A template in the `go template` format with the `helm` functions that renders objects into the namespaces of a project, next to the objects that the structured fields render.

  The template gets the project name in `{{ .projectName }}`, the template name in `{{ .projectTemplateName }}` and the project parameters completed with the defaults of `parametersSchema` in `{{ .parameters }}`, the same values as `resourcesTemplate`. A project whose parameters turn into objects or keys of the rendered manifests instead of values is refused, so quote every substitution.

  The controller applies the objects with server-side apply as the `system:multitenancy-manager:project:<PROJECT NAME>` user, and the `d8-manifests` ProjectRoleBinding of the project grants this user the `d8:project:admin` role. So an object gets only what this role allows in the namespaces of the project, not the rights of the controller. An AuthorizationRule of the `Admin` access level is refused when the manifests create it or raise an existing one to this level, although the role allows it. The administrators of a project are set in `.spec.administrators` of the Project. The objects are applied once the binding is Ready, and the `ManifestsApplied` condition of the project says whether they are applied and names the objects that the API server refused with the code of each refusal, such as 403.

  An object without `metadata.namespace` is created in the main namespace of the project, and a namespace that an object names has to be a namespace of the project or one that a ProjectNamespace of the same `manifests` makes. The objects of such a namespace are applied once it exists, and until then the `ManifestsApplied` condition names the namespaces they wait for. See Additional namespaces in manifests. A cluster-scoped object, an object in a namespace outside the project and an object with the `heritage: multitenancy-manager` or `heritage: deckhouse` label are refused. The objects get the `projects.deckhouse.io/project` label with the project name and the `projects.deckhouse.io/source: manifests` label, and `.status.manifestObjects` of the project lists them. An object that the template does not render any more is deleted.

  An object that a switch of the project from a Helm text left in place and listed in `.status.keptObjects` is taken over when `manifests` renders an object of the same kind,

namespace and name. The object then has what `manifests` renders, and a field that the Helm text set and `manifests` leaves out is removed, the annotations of Helm included. The same list moves an object from the release of the structured fields to `manifests`. When the structured fields stop rendering an object that `manifests` render under the same kind, namespace and name, the controller leaves it in place with the `projects.deckhouse.io/kept-from-helm-template: "true"` label and lists it in `.status.keptObjects`, and `manifests` take it over in the same change. If `manifests` cannot apply it, the object stays on the list. See Moving an object of the release to manifests. Any other object that exists already and that `manifests` did not create is changed only when it has the two labels, and otherwise the `ManifestsApplied` condition names it. An object that `manifests` created and that the template renders otherwise after a change, for example a NetworkPolicy named `isolated` once `networkPolicy.mode` is `Isolated`, goes to the objects the project renders from the template in the same change. It gets what the template renders, and a field that only `manifests` set is removed.

Like the objects that the structured fields render, these objects are changed through the template only. The `d8-multitenancy-manager-manifests-objects` admission policy refuses to create, change or delete an object with the `projects.deckhouse.io/source: manifests` label to everyone, project and cluster administrators included. Only the user the `manifests` of the project are applied as, the controller, the service accounts of the `d8-*` namespaces and the cluster components can do it. Anyone who may update a workload may restart its rollout, when the `kubectl.kubernetes.io/restartedAt` annotation of its pod template is the only change. `metadata.resourceVersion`, `metadata.generation` and `metadata.managedFields` are not compared, because the API server sets them on the write. The controller applies the objects again after a while, which puts back the fields that the template sets if something else changed them. A project template can be read by users who have no access to its projects, the ones with the `User` access level included, so do not put secret values into `manifests`.

The field cannot be set together with `resourcesTemplate`. `deckhouse.io/v1alpha1` has no place for it, so a write of a template that has it through `deckhouse.io/v1alpha1` is refused. See Describing other objects in manifests.

- **spec.namespaceMetadata**
  object

  Extra labels and annotations for the project namespaces.

  - **spec.namespaceMetadata.annotations**

    A literal annotation map or a parameter reference `{fromParam: "<name>"}`.

    The annotations are set on the main namespace of the project only.

  - **spec.namespaceMetadata.labels**

    A literal label map or a parameter reference `{fromParam: "<name>"}`.

    The labels are set on every namespace of the project, including the additional ones, and are listed in the `projects.deckhouse.io/template-labels` annotation of each.

    Labels that the module manages (`heritage`, `app.kubernetes.io/managed-by`, `projects.deckhouse.io/*`, `multitenancy.deckhouse.io/*`, and the pod security, monitoring and scanning labels) and the `kubernetes.io/metadata.name` label cannot be set in this field. A template that sets one here is refused, and so is a Project whose parameters put one here. A template or Project that already has such a label can still be changed while it keeps the label as it is. The label is not applied, and the response warns about it.

    GitOps tracking labels (`app.kubernetes.io/instance`, `argocd.argoproj.io/*`, `kustomize.toolkit.fluxcd.io/*`, `helm.toolkit.fluxcd.io/*`, `kapp.k14s.io/*`) are set on the main namespace only. A custom tracking key is inherited like any other label.

    When a namespace becomes a project automatically, its own labels are not copied into the Project and do not reach this field. A label declared in `spec.parameters.namespace.labels` of the Project reaches it through the built-in templates and is inherited.

- **spec.networkPolicy**
  object

  Network isolation of the project namespaces.

  - **spec.networkPolicy.mode**

    Network isolation mode of the project namespaces.

    - `Isolated`: Every namespace of the project gets the `isolated` NetworkPolicy. It allows incoming traffic only from the project namespaces, Prometheus, the ingress-nginx controller and the service-with-healthchecks agent, and outgoing traffic only to the project namespaces and to DNS in `kube-system`.
    - `NotRestricted`: No NetworkPolicy is created.

If the field is not set, no NetworkPolicy is created either.

A literal string (`Isolated`/`NotRestricted`) or a parameter reference `{fromParam: "<name>"}`. The webhook refuses a literal outside the two values.

- **spec.nodeSelector**

  Node selector of the pods in the main namespace of the project. A literal label map or a parameter reference `{fromParam: "<name>"}`.

  It is set as the `scheduler.alpha.kubernetes.io/node-selector` annotation of the main namespace and is merged into the node selector of every pod there. A pod whose selector conflicts with it is rejected. The additional namespaces of the project do not get it.

- **spec.parametersSchema**
  object

  OpenAPI v3 schema of the parameters of a project.

  A boolean `requiredRequests` parameter declared here is read by name: when it is `true` for a project, the pods in every namespace of the project must have CPU and memory requests (an OperationPolicy in the `Deny` mode).

  - **spec.parametersSchema.openAPIV3Schema**
    object

    Required value

    OpenAPI specification that the parameters of a project are validated against when the project is created or changed.

- **spec.podSecurityStandard**

  Pod Security Standard profile applied to the project namespaces.

  - `Privileged`: Unrestricted policy.
  - `Baseline`: Minimally restrictive policy that prevents known privilege escalations.
  - `Restricted`: Heavily restricted policy that follows current pod hardening best practices.

  A literal string (`Privileged`/`Baseline`/`Restricted`) or a parameter reference `{fromParam: "<name>"}` pointing at a `parametersSchema` property. The webhook refuses a literal outside the three values and a reference to an undeclared or mistyped parameter.

- **spec.resources**
  array of objects

  Rules of access to cluster-wide resources for the projects of the template.

  The controller turns them into the `template-<TEMPLATE_NAME>-inline` ClusterResourceGrantPolicy, which applies to the namespaces of the projects of this template.

  Access settings for one type of cluster-wide resource.

  - **spec.resources.allowed**
    array of strings

    List of names of cluster-wide resources available to the projects.

    Combined with the resources selected with `allowedSelector`.

  - **spec.resources.allowedSelector**
    object

    Label selector of cluster-wide resources available to the projects.

    Combined with the resources listed in `allowed`. Applies only to cluster-wide resources represented by Kubernetes objects.

    - **spec.resources.allowedSelector.matchExpressions**
      array of objects

      List of label selector expressions.

      All expressions must match.

      - **spec.resources.allowedSelector.matchExpressions.key**
        string

        Required value

        Label key to which the expression applies.

      - **spec.resources.allowedSelector.matchExpressions.operator**

string

Required value

Operator that defines the relationship between the key and the set of values.

- **spec.resources.allowedSelector.matchExpressions.values**
  array of strings

  List of label values.

- **spec.resources.allowedSelector.matchLabels**
  object

  Set of labels of cluster-wide resources.

  All specified labels must match.

- **spec.resources.availabilityDefault**
  string

  Availability of the cluster-wide resources of this type that `allowed`, `allowedSelector`, `denied` and `deniedSelector` do not name.

  Overrides `defaultAvailability` of the GrantableClusterResourceDefinition. If `allowed` or `allowedSelector` is set and this field is not, all other resources of this type are unavailable to the projects.

  Allowed values: `All`, `None`

- **spec.resources.default**
  string

  Name of the cluster-wide resource used by default in the projects.

  Overrides the value determined with `defaultFrom` in the GrantableClusterResourceDefinition.

- **spec.resources.denied**
  array of strings

  List of names of cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`.

- **spec.resources.deniedSelector**
  object

  Label selector of cluster-wide resources unavailable to the projects.

  Deny rules take precedence over `allowed` and `allowedSelector`.

  - **spec.resources.deniedSelector.matchExpressions**
    array of objects

    List of label selector expressions.

    All expressions must match.

    - **spec.resources.deniedSelector.matchExpressions.key**
      string

      Required value

      Label key to which the expression applies.

    - **spec.resources.deniedSelector.matchExpressions.operator**
      string

      Required value

      Operator that defines the relationship between the key and the set of values.

    - **spec.resources.deniedSelector.matchExpressions.values**
      array of strings

      List of label values.

  - **spec.resources.deniedSelector.matchLabels**
    object

    Set of labels of cluster-wide resources.

All specified labels must match.

- **spec.resources.resourceName**
  string

  Required value

  Name of the GrantableClusterResourceDefinition that registers the type of cluster-wide resource.

- **spec.resourcesTemplate**
  string

  A template in the `go template` format with the `helm` functions that renders the objects of a project.

  The field is deprecated and is kept for the templates created through `deckhouse.io/v1alpha1`. Describe new templates with the structured fields and `manifests`. No template can get a new `resourcesTemplate` through either version: a create with a non-empty one and an update that sets one on a template that has none are refused, and a create through `deckhouse.io/v1alpha1` is refused as a whole. A template that has one keeps it, and it can be changed through either version while `deckhouse.io/v1alpha1` can describe the whole template. So a template with a Helm text cannot be created again, for example from a backup or from Git; move it to the structured fields and `manifests` first. The field leaves `deckhouse.io/v1alpha2` together with `deckhouse.io/v1alpha1` in a future release. See Moving a template from a Helm text.

  When the field is not empty, the template is rendered from it. It cannot be set together with the fields that render objects into the project namespaces (`podSecurityStandard`, `networkPolicy`, `namespaceMetadata`, `manifests` and the others), while `resources` and `grantPolicies` apply in either case. A template that projects use keeps a non-empty `resourcesTemplate` until a change sets such fields or `manifests` in its place.

  After such a change, as after a move of a project to a structured template, the AuthorizationRule and ResourceQuota objects of the Helm text that `.spec.administrators` and `.spec.quota` of the project replace are deleted once these fields work, and an AuthorizationRule that names nobody from `.spec.administrators` any more is deleted at once. Every other object of the Helm text that the new fields do not render under the same kind and name is left in place and listed in `.status.keptObjects` of the project, until `manifests` renders an object of the same kind, namespace and name and takes it over. `.status.keptObjects` also lists an object of the release of the structured fields that `manifests` render when the release drops it, until `manifests` take it over. See Switching a project from a Helm text and Moving an object of the release to manifests.

  The template gets the project name in `{{ .projectName }}`, the template name in `{{ .projectTemplateName }}` and the project parameters completed with the defaults of `parametersSchema` in `{{ .parameters }}`. When `parametersSchema` declares the `administrators` and `resourceQuota` parameters, they hold the administrators and the quota of the project, laid out as in a `deckhouse.io/v1alpha2` Project. A project whose parameters turn into objects or keys of the rendered manifests instead of values is refused, so quote every substitution.

  An object without `metadata.namespace` or in a namespace outside the project is created in the main namespace of the project. The labels the template puts on the main namespace, except the ones the module manages and the GitOps tracking labels, are listed in its `projects.deckhouse.io/template-labels` annotation and are set on the additional namespaces too.

- **spec.runtimeAudit**
  object

  Runtime audit rules; effective only when allowedUIDs/allowedGIDs are set.

  - **spec.runtimeAudit.enabled**

    A literal boolean or a parameter reference `{fromParam: "<name>"}`.

- **spec.title**
  string

  Short human-readable name of the template.

- **spec.tolerations**

  Default tolerations of the pods in the main namespace of the project. A literal list of tolerations or a parameter reference `{fromParam: "<name>"}`.

  They are set as the `scheduler.alpha.kubernetes.io/defaultTolerations` annotation of the main namespace. The additional namespaces of the project do not get them.

- **status**
  object

    - **status.message**
      string

      Message indicating the cause of the current status.

    - **status.ready**
      boolean

      Whether the template is ready to use. Indicates that the template has been successfully validated.

Deprecated resource. Support for the resource might be removed in a later release.

Describes the project template.

This resource is referenced by the Project custom resource.

The version is deprecated. A template is created through `deckhouse.io/v1alpha2`, and a create through `deckhouse.io/v1alpha1` is refused. The templates created through `deckhouse.io/v1alpha1` earlier are read, changed and deleted through it as before.

- **spec**
  object
    - **spec.description**
      string

      Arbitrary description of the template's purpose. Specify an empty string if no description is required.

    - **spec.parametersSchema**
      object

      Required value

      Schema of the project parameters.

      Only `openAPIV3Schema` is supported.

        - **spec.parametersSchema.openAPIV3Schema**
          object

          Required value

          OpenAPI specification for parameters.

          This is used to check the validity of the parameters that are passed when creating a new project.

    - **spec.resourcesTemplate**
      string

      Required value

      A template that defines resources in the `go template` format to be created when setting up a new project.

      Templates are fully compatible with all `helm` functions.

      See Moving a template from a Helm text.

      The following parameters are available in templates:

        - `{{ .projectName }}` — the name of the `Project` for which the template is being rendered.
        - `{{ .projectTemplateName }}` — the name of the `ProjectTemplate`.
        - `{{ .parameters }}` — a dictionary of custom values, described in the .spec.parametersSchema and defined in the .spec.parameters.

      `.metadata.namespace` of an object is optional: an object without it is created in the main namespace of the project.

- **status**
  object
    - **status.message**
      string

      Message indicating the cause of the current status.

    - **status.ready**
      boolean

      Whether the template is ready to use. Indicates that the template has been successfully validated.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The multitenancy-manager module: usage examples

> ⚠ The module lifecycle stage: Preview

Default project templates

Deckhouse Platform includes the following project templates. They are cumulative: each one includes the capabilities of the previous one and adds its own. Parameter values are set in the `.spec.parameters` field of a Project:

- `simple`: Minimal template that creates only the project namespace. Use it when you only need an isolated namespace managed as a project and configure access and limits through the standard fields and project role bindings.

  Parameters:

  - `namespace.labels` and `namespace.annotations`: Extra labels and annotations for the project namespace.
  - `requiredRequests` (default `false`): When `true`, the pods in the project must specify CPU and memory requests (an OperationPolicy in the `Deny` mode).

- `default`: Template that covers basic project use cases. On top of the namespace, it sets up network isolation, a pod security profile, extended monitoring and log shipping.

  Parameters (besides the ones listed for `simple`):

  - `networkPolicy`: `Isolated` (default) allows incoming traffic only from the project namespaces, Prometheus, the ingress-nginx controller and the service-with-healthchecks agent, and outgoing traffic only to the project namespaces and to DNS in `kube-system`. `NotRestricted` allows all traffic.
  - `podSecurityProfile`: The Pod Security Standards profile for the project namespaces: `Baseline` (default) prevents known privilege escalations, `Restricted` applies the strictest hardening practices, `Privileged` restricts nothing. An empty string sets no profile. The project puts no `security.deckhouse.io/pod-policy` label on its namespaces, and the default policy of the `admission-policy-engine` module applies to them.
  - `extendedMonitoringEnabled` (default `true`): Alerts on controller outages and restarts, 5xx errors in ingress-nginx and low free space on the project's persistent volumes.
  - `clusterLogDestinationName`: The name of the ClusterLogDestination to ship the project logs to. Left unset, the project logs are not shipped anywhere.
  - `requiredRequests` (default `true` in this template): When `true`, the pods in the project must specify CPU and memory requests (an OperationPolicy in the `Deny` mode). When an existing namespace becomes a project, the parameter is set to `false`, so running workloads are not blocked.

- `secure`: Includes all the capabilities of the `default` template, and additionally restricts the users and groups inside containers, audits their calls to the kernel and scans images for vulnerabilities.

  Parameters (besides the ones listed for `default`):

  - `allowedUIDs` and `allowedGIDs`: The ranges (`min`, `max`) of IDs permitted for users and groups inside the project containers. See Security Context.
  - `runtimeAuditEnabled` (default `false`): Audit rules that catch calls to the kernel and find malicious activity. They work only if the UID/GID ranges are set.
  - `securityScanningEnabled` (default `true`): Periodic scanning of the launched images for known vulnerabilities (CVE) with Trivy, every 24 hours.

- `secure-with-dedicated-nodes`: Includes all the capabilities of the `secure` template, and additionally places the pods of the main namespace of the project on dedicated nodes.

  Parameters (besides the ones listed for `secure`). If `dedicatedNodes` is set, at least one of its two fields must be set. Without `dedicatedNodes`, the pods are not placed on dedicated nodes:

- `dedicatedNodes.nodeSelector`: The node selector of the main namespace. It is merged into the node selector of every pod there, and a pod whose selector conflicts with it is rejected.
- `dedicatedNodes.defaultTolerations`: Tolerations in the format of the pod's `spec.tolerations`. They are **added** to the pods created in the main namespace.

The `default`, `secure`, and `secure-with-dedicated-nodes` templates are described in the structured form (`deckhouse.io/v1alpha2`). The `simple` template is a minimal structured template that creates the namespace, sets its labels and annotations from the project parameters and, with `requiredRequests: true`, creates the OperationPolicy.

The exact set of parameters is in the template installed in the cluster. To view the parameter schema of a template, use the command:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o
jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

To view the whole template, use the command:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o yaml
```

Creating a project

To create a project, follow these steps:

1. Create a Project, specifying the name of the project template in .spec.projectTemplateName field.
2. Set the standard fields .spec.administrators and .spec.quota. They work with any template.
3. In the .spec.parameters field of the Project resource, specify the parameter values suitable for the ProjectTemplate .spec.parametersSchema.openAPIV3Schema.

   Example of a Project created from the `default` ProjectTemplate:

   ```
   apiVersion: deckhouse.io/v1alpha3
   kind: Project
   metadata:
     name: my-project
   spec:
     description: This is an example from the Deckhouse documentation.
     projectTemplateName: default
     # Standard fields, managed by the Project resource itself.
     administrators:
       - kind: Group
         name: k8s-admins
     quota:
       requests.cpu: "5"
       requests.memory: 5Gi
       requests.storage: 1Gi
       limits.cpu: "5"
       limits.memory: 5Gi
     # Template-specific parameters.
     parameters:
       networkPolicy: Isolated
       podSecurityProfile: Restricted
       extendedMonitoringEnabled: true
   ```

   The Project API versions are described in Project API versions.
4. To check the status of the project, execute the command:

   ```
   d8 k get projects my-project
   ```

   A successfully created project should be in the `Deployed` state. If the state equals `Error`, add the `-o yaml` argument to the command (e.g., `d8 k get projects my-project -o yaml`) to get more detailed information about the error.

Project API versions

The Project API is served as `deckhouse.io/v1alpha3`. `v1alpha2` manifests keep working: a conversion webhook lifts `parameters.administrators` and `parameters.resourceQuota` into the `.spec.administrators` and `.spec.quota` standard fields automatically. If one administrator does not fit `.spec.administrators` (for example, it has `accessLevel`), the whole list stays in `parameters` and `.spec.administrators` is not filled. The same holds for a quota with anything other than quantities (for example, `scopes`), for a quota with the number `0` anywhere in it, and for an empty list or quota. A template that checks a value with `with` skips the number `0` and sets no limit, while in `.spec.quota` it would be a hard limit of zero. The string `"0"` is lifted into `.spec.quota` as such a limit. A `v1alpha2` client reads a `0` from `.spec.quota` as the string `"0"`. A project cannot set `.spec.administrators` together with `parameters.administrators`, or `.spec.quota` together with `parameters.resourceQuota`. The `spec.grants` field exists only in `deckhouse.io/v1alpha3`: a change of a project that has it through `deckhouse.io/v1alpha2` is refused, see Granting cluster-wide resources to one project.

The `deckhouse.io/v1alpha1` version is no longer served, and the API server refuses a manifest with `apiVersion: deckhouse.io/v1alpha1`. Before applying such a manifest, change `apiVersion` to `deckhouse.io/v1alpha3`, rename `spec.projectTypeName` to `spec.projectTemplateName` (a ProjectTemplate with that name must exist) and `spec.template` to `spec.parameters`, and move `administrators` and `resourceQuota` from the parameters to `.spec.administrators` and `.spec.quota`.

Creating a project without specifying a template

The `projectTemplateName` field is optional: when omitted, the `simple` template is used. Such a project consists only of the namespace and the standard fields (administrators, quota) — no template policies are created in it. Use it when the namespace needs no template settings or they are managed by other means:

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-plain-project
spec:
  administrators:
    - kind: Group
      name: k8s-admins
  quota:
    requests.cpu: "2"
```

A template can be assigned later by setting it in `.spec.projectTemplateName`.

Project naming rules

The project name is also the name of its main namespace, so the following rules are checked when a project is created:

- The name cannot start with `d8-` or `kube-`: these prefixes are reserved for system namespaces.
- The name cannot be `default` or `deckhouse`: these names belong to the virtual projects.
- The name cannot be longer than 61 characters.
- The name cannot be the name of an existing namespace, or of a namespace that a ProjectNamespace has claimed. The resulting name `<project>-<spec.name>` is taken from the moment the ProjectNamespace is created, before the controller creates the namespace. A namespace that belongs to no project becomes a project automatically.
- The name cannot be the name of a namespace whose ServiceAccounts are still named among the subjects of a ClusterProjectRoleBinding, or of a ProjectRoleBinding of a project that the name extends with a dash and a suffix (`foo` for `foo-extra`). Such subjects are often left from a deleted additional namespace, and without this rule the ServiceAccounts of the new project would get the rights of that binding. Remove these subjects from the binding first. The refusal does not name the binding.

A project can be named after another project with a dash and a suffix (`foo-bar` beside `foo`). It is a separate project with a namespace of its own, not an additional namespace of `foo`, and the template, administrators, quota and bindings of `foo` do not apply to it. The webhook allows such a project with a warning, and the controller sends the `NamedLikeAdditionalNamespace` Warning event on the new project. The `MultitenancyManagerProjectNamedLikeAdditionalNamespace` alert lists every such pair of projects, including the ones made from namespaces that existed before. To add a namespace to a project, create a ProjectNamespace.

Project status and diagnostics

The `.status.state` field of a project is either `Deployed` (all project resources are in sync, or the objects of `manifests` wait for their binding) or `Error`. The cause of an error is described in the conditions (`.status.conditions`):

```
d8 k get project my-project -o jsonpath='{range .status.conditions[*]}{.type}={.status}:
{.message}{"\n"}{end}'
```

Condition	False means
`ProjectTemplateFound`	The template referenced in `.spec.projectTemplateName` was not found.
`Validated`	The project parameters failed validation against the template schema (`parametersSchema`).
`ResourcesUpgraded`	The project resources could not be created or updated from the template (details in `message`).
`StandardFieldsApplied`	The standard fields (quota or administrators) could not be applied.
`TemplateRolesAllowed`	The template creates a binding to a role forbidden for granting in projects — the project switches to `Error`, the role is named in `message`.
`ManifestsApplied`	The objects of the template's `manifests` are not applied. The `message` says what they wait for, why none of them is applied, which existing objects they leave alone, or which objects the API server refused, with the code of each refusal (for example, `403 Forbidden`).
`HelmOwnership`	The namespace of the project belongs to another Helm release. The `message` names the release (see Creating a project automatically for a namespace).
`NamespaceDeleted`	The project is being deleted, and its namespace is still terminating. The `message` says since when and what remains in it.
`QuotaWithinLimits`	The usage of the project is above its quota; the `message` names the limits. A project without a quota does not have the condition.

The `HandoverPending` condition is read the other way round. It is `True` while AuthorizationRule and ResourceQuota objects kept from a Helm text wait for the standard fields that replace them, and it is removed once they are deleted (see Switching a project from a Helm text).

Other useful status fields:

- `.status.namespaces`: All namespaces of the project with their kind (`Main` or `Additional`).
- `.status.usage`: The usage of the project summed over all of its namespaces, taken from its ProjectQuotaStatus.
- `.status.resources`: State of the individual objects created from the template.
- `.status.keptObjects`: Objects of a Helm text and of the release of the structured fields that the release left in place instead of deleting them, until `manifests` take them over.
- `.status.manifestObjects`: Objects applied from `manifests` of the template.

Service objects of a project

The controller creates service objects in the project namespaces. They are managed automatically — editing them manually is not possible (the attempt is rejected):

Object	Where	Comes from
`ProjectQuotaStatus/<PROJECT_NAME>`	Cluster-scoped	The `.spec.quota` field of the project (see Project quota).

Object	Where	Comes from
`ResourceQuota/d8-project-quota`	Every namespace of the project	The `.spec.quota` field of the project.
`ProjectRoleBinding/d8-administrators`	The main namespace	The `.spec.administrators` field of the project.
`ProjectRoleBinding/d8-manifests`	The main namespace	The `manifests` field of the project template. It grants the `d8:project:admin` role to the user that applies these objects.
`RoleBinding/d8:prb:<name>`	Every namespace of the project	The fan-out of the ProjectRoleBinding named `<name>`.
`RoleBinding/d8:cprb:<name>`	Every namespace of every project	The fan-out of the ClusterProjectRoleBinding named `<name>`.

When the source object (a binding, the quota field, etc.) is removed, the corresponding service objects are removed automatically.

Project creator and author of the last change

The `projects.deckhouse.io/created-by` annotation of a project holds the name of the user who created it. The `projects.deckhouse.io/last-modified-by` annotation holds the name of the user who last changed its spec, labels or annotations.

If the webhook is unavailable, the author annotations may be missing or contain outdated values.

To see both, run:

```
d8 k get project my-project -o
jsonpath='{.metadata.annotations.projects\.deckhouse\.io/created-by}{"\n"}
{.metadata.annotations.projects\.deckhouse\.io/last-modified-by}{"\n"}'
```

Virtual projects

Besides the user-created projects, the `d8 k get projects` list always contains two **virtual** projects (labeled `projects.deckhouse.io/virtual-project: "true"`):

- `deckhouse` — groups system namespaces (`d8-*`, `kube-*`, `upmeter-*`, `heritage: deckhouse` / `heritage: upmeter`);
- `default` — groups remaining namespaces that do not belong to any project (the `default` namespace itself).

Virtual-project status is rebuilt from the live namespace list: a deleted namespace disappears from that list. Virtual projects do not recreate namespaces.

Virtual projects exist for completeness: with them, every namespace of the cluster belongs to some project. They cannot be managed: they are not editable, ProjectNamespace and ProjectRoleBinding resources cannot be created in them, and ClusterProjectRoleBinding does not extend to them. Their project template, `virtual`, is reserved for them: a user project with `projectTemplateName: virtual` is refused by the webhook, since such a project would get no namespace of its own. The `projects.deckhouse.io/virtual-project: "true"` label is reserved too: the webhook refuses a project that is created with it or gets it added. A project is virtual by its template, not by the label, so a user project that already carries the label is an ordinary project; it keeps the label with a warning until the label is removed. No ClusterResourceGrantPolicy applies to their namespaces either, even one whose selectors match a virtual Project, because the cluster resource availability rules apply only in namespaces with the `projects.deckhouse.io/project` label, and the namespaces of virtual projects do not carry it.

No ProjectTemplate object named `virtual` exists, and the webhook refuses to create one. A template with this name that is already in the cluster is not used by any project and can be deleted.

Additional project namespaces

In Deckhouse Platform Open (CE), a project has only its main namespace.

If an application needs several namespaces (for example, a separate one for a cache or queues), add

them to the project with the ProjectNamespace resource. A ProjectNamespace is created **in the main namespace of the project**; the resulting namespace is named `<project name>-<spec.name>`:

```
apiVersion: deckhouse.io/v1alpha3
kind: ProjectNamespace
metadata:
  name: cache
  namespace: my-project
spec:
  name: cache   # The my-project-cache namespace will be created.
```

You can check the project composition in its status:

```
d8 k get project my-project -o jsonpath='{.status.namespaces}'
```

The rules for working with ProjectNamespace:

- The `spec.name` field is immutable: to rename a namespace, delete the ProjectNamespace and create a new one.
- The resulting name `<project name>-<spec.name>` cannot be longer than 63 characters (the Kubernetes limit on namespace names).
- A ProjectNamespace can only be created in the main namespace of a project. It cannot be “nested” into an additional namespace or a foreign project.
- The resulting name must be free. The request is rejected when a namespace with that name exists and is not a namespace of this project, when a project has that name, or when another ProjectNamespace already claims it (the ProjectNamespace `a-backend` of project `team` and the ProjectNamespace `backend` of project `team-a` both claim `team-a-backend`).
- The request is also rejected when the ServiceAccounts of a namespace with the resulting name are still named among the subjects of a ClusterProjectRoleBinding, or of a ProjectRoleBinding of another project that the resulting name extends with a dash and a suffix (`team` for `team-a-backend` of project `team-a`). Remove these subjects from the binding first. A ProjectRoleBinding of the same project is no obstacle, and the ServiceAccounts it names get the rights back once the namespace is created again.
- Deleting a ProjectNamespace deletes its namespace. Deleting the project deletes all of its namespaces.
- A namespace with the resulting name that is not a namespace of this project is never taken into it, even when it appears after the ProjectNamespace was created. The ProjectNamespace then does not become ready, and deleting it leaves the namespace as it is.
- A project can be deleted only by a user allowed to delete each of its namespaces (the `delete` verb on `namespaces`): the main namespace and every namespace labeled `projects.deckhouse.io/project: <project name>`. Otherwise the request is refused, and the refusal names the namespace the user may not delete. A namespace that does not exist is not checked. The identities the Project webhook skips (see Who bypasses admission) delete projects without this check.

What applies to the additional namespaces

The following automatically applies in **all** namespaces of the project (the main and the additional ones alike):

- **Access**: the ProjectRoleBinding and ClusterProjectRoleBinding bindings, including the automatic access of the project administrators. When a new namespace is added, all existing bindings fan out into it without any user action.
- **Namespaced template objects**: the network policy (`networkPolicy.mode: Isolated`) and the log collection setup (`logShipping`) are created in every namespace of the project. The network isolation allows traffic between the namespaces of one project. A Helm `resourcesTemplate` creates its objects in the main namespace unless an object names another namespace of the project. So does `manifests`, whose objects are not applied while one of them names a namespace outside the project.
- **Cluster-scoped template policies** (`OperationPolicy`, the `SecurityPolicy` from `allowedUIDs`/`allowedGIDs`): they select namespaces by the `projects.deckhouse.io/project` label, that is, they cover the whole project.
- **The project quota**: the `d8-project-quota` ResourceQuota with the limits of the project quota is created in every namespace of the project.
- **Inherited labels**: the pod security profile (`security.deckhouse.io/pod-policy`), extended monitoring (`extended-monitoring.deckhouse.io/enabled`), vulnerability scanning (`security-scanning.deckhouse.io/enabled`), the template label (`projects.deckhouse.io/project-template`) and the labels the template sets through `namespaceMetadata.labels` are synced from the main namespace to the additional ones. For a template with `resourcesTemplate`, these

are the labels its Helm text puts on the main namespace. The sync is complete: if a feature is turned off or a label is removed in the template, the label is removed from the additional namespaces as well. The labels of the template are listed in the `projects.deckhouse.io/template-labels` annotation of each project namespace; they are changed through the Project or its ProjectTemplate, and a direct edit of one on an additional namespace is reverted. Thanks to the template label, the cluster resource availability rules also apply in all namespaces of the project.

The following stays in the **main** namespace only:

- The annotations from the template's `namespaceMetadata`
- The labels and annotations set on the main namespace directly or by a GitOps tool, including the ones the namespace had before it became a project, other than the pod security, monitoring and scanning labels above
- The GitOps tracking labels (`app.kubernetes.io/instance`, `argocd.argoproj.io/*`, `kustomize.toolkit.fluxcd.io/*`, `helm.toolkit.fluxcd.io/*`, `kapp.k14s.io/*`), even when the template sets them
- The node placement annotations (from the template's `nodeSelector` and `tolerations` fields)

GitOps tracking labels are not inherited because a GitOps tool would treat an additional namespace that carries them as its own and could delete it. The module recognizes only the keys listed above. A custom tracking key that the template sets, for example the one configured with `application.instanceLabelKey` in Argo CD, is inherited.

When a namespace becomes a project automatically, its own labels and annotations stay on it and are not copied into the Project, so the additional namespaces of the project do not inherit them. The pod security, monitoring and scanning labels are the exception. Adoption turns them into parameters of the project, and they are inherited as on any project. To put another label of such a namespace on the additional namespaces too, declare it in `spec.parameters.namespace.labels` of the Project, or set it on each additional namespace directly.

The built-in templates of adopted projects are rewritten by the controller, so a label added to their `namespaceMetadata.labels` does not stay there. A grant policy whose `namespaceSelector` matches such a label does not cover the new additional namespaces until the label is declared in the Project. To cover the whole project, select it with `projectSelector`.

Labels of the project namespaces

Label	Main	Additional	Purpose
`projects.deckhouse.io/project: <PROJECT_NAME>`	✓	✓	Project ownership — the common label of all namespaces of the project
`projects.deckhouse.io/project-namespace: <spec.name>`	—	✓	Marks an additional namespace (the name of the ProjectNamespace resource)
`projects.deckhouse.io/project-template: <PROJECT_TEMPLATE_NAME>`	✓	✓	The project template; the cluster resource availability rules match by it
`heritage: multitenancy-manager`	✓	✓	The namespace is managed by the project controller: its `spec`, finalizers and the labels listed in this table are changed through the Project; other labels and annotations may be changed directly
`projects.deckhouse.io/quota: pending`, `projects.deckhouse.io/quota: enforced`	✓	✓	The namespace belongs to a project with a quota; only the controller sets the label

Label	Main	Additional	Purpose
`quota.projects.deckhouse.io/vm: enforced`, `quota.projects.deckhouse.io/lb: enforced`	✓	✓	The quota of the project has keys of virtual machines or of load balancers
`security.deckhouse.io/pod-policy`, `extended-monitoring.deckhouse.io/enabled`, `security-scanning.deckhouse.io/enabled`	✓	✓ (inherited)	Policies and features from the project template
Labels from the template's `namespaceMetadata.labels`	✓	✓ (inherited)	Labels of the project template. They are listed in the `projects.deckhouse.io/template-labels` annotation of the namespace and changed through the Project or its ProjectTemplate
Any other label set on a namespace directly or by a GitOps tool, including the labels a namespace had before it became a project	—	—	Not set by the module. The project release neither reverts nor restores it, and the additional namespaces do not inherit it

The common `projects.deckhouse.io/project` label makes it possible to select the project namespaces with a plain `get ns`. To get all namespaces of the project (main and additional), use the command:

```
d8 k get ns -l projects.deckhouse.io/project=my-project
```

To get additional namespaces only:

```
d8 k get ns -l 'projects.deckhouse.io/project=my-
project,projects.deckhouse.io/project-namespace'
```

To get the main namespace only:

```
d8 k get ns -l 'projects.deckhouse.io/project=my-
project,!projects.deckhouse.io/project-namespace'
```

Creating a project automatically for a namespace

A namespace created directly (for example, `d8 k create ns my-app`) becomes a project with the same name:

- the template is picked from what the namespace already carries: `secure` if it has the `security-scanning.deckhouse.io/enabled` label, `default` if it has `security.deckhouse.io/pod-policy` or `extended-monitoring.deckhouse.io/enabled`, and `simple` otherwise;
- the project parameters are filled in from the current state of the namespace, so nothing inside it changes: the network policy stays unrestricted and the Pod Security Standard keeps the value the namespace already had;
- from then on the project is the source of truth and is edited like any other project. The namespace of the project cannot be deleted directly: the request is refused with the `Manual modification is forbidden` message. To remove the environment, delete the Project: `d8 k delete project <NAME>` deletes the namespace with it and returns once the namespace is gone. Only a user allowed to delete the namespace can delete its Project (see Additional project namespaces).

Users can create a namespace directly only while the `allowNamespacesWithoutProjects` parameter is `true` (the default). With `false`, only cluster administrators (`system:masters`, `system:sudouser`), Deckhouse, the module controller and system components create namespaces directly (see Who bypasses admission). Other users create a Project or a ProjectNamespace instead.

Every such namespace becomes a project of its own, whatever its name. Of the namespaces `app` and `app-staging`, `app-staging` becomes a separate project and not an additional namespace of `app`. The controller creates the project, so no admission warning reaches the author of the namespace. Instead, the controller sends the `NamedLikeAdditionalNamespace` Warning event on the new project and on the namespace. The events of cluster-scoped objects are kept in the `default` namespace, so reading them needs access to the events there: `d8 k -n default get events --field-selector reason=NamedLikeAdditionalNamespace`. The `MultitenancyManagerProjectNamedLikeAdditionalNamespace` alert lists such pairs, see Project naming rules.

System namespaces (`d8-*`, `kube-*`, `upmeter-*`, `default`, and anything labeled `heritage: deckhouse` or `heritage: upmeter`) are never adopted: they are listed on the virtual `deckhouse` project (except `default`, which stays on the virtual `default` project). No label leaves a new user namespace without a project: users cannot put the `heritage: upmeter` label on a namespace, and only the module sets the `projects.deckhouse.io/project` label. A namespace whose name is longer than 61 characters (the Project name limit) is also skipped, and so is a namespace named `deckhouse`: a Project of that name is the virtual project.

So that no namespace stays outside a project, the `d8-multitenancy-manager-namespace-adoption` ValidatingAdmissionPolicy refuses to create a namespace whose name is longer than 61 characters, starts with `upmeter-` or is `deckhouse`, and to put the `heritage: upmeter` label on a namespace. The policy works whatever `allowNamespacesWithoutProjects` says and lets through the users and groups listed for the `d8-multitenancy-manager-namespace-creation` policy in Who bypasses admission. A namespace that stays without a project for 10 minutes, for example while no replica of the module controller leads, raises the `D8NamespaceNotAdopted` alert. The controller pods export the metric of the alert, so the alert is not raised while none of the pods is `Ready`.

The policy does not change the namespaces that exist already, so a namespace that was created before it, with one of these names or with the `heritage: upmeter` label, stays outside every project. To list the namespaces that belong to no project and are not system ones, run:

```
d8 k get namespaces -l '!projects.deckhouse.io/project,heritage!=deckhouse' -o name | grep -Ev '^namespace/(d8-|kube-|default$|upmeter-probe-namespace-)'
```

The list holds the namespaces that get no project: the names longer than 61 characters, `deckhouse`, the other `upmeter-*` names and the namespaces with the `heritage: upmeter` label. The namespaces of the upmeter agent can be among them, and they must not be changed. To bring a namespace under a project, remove the `heritage: upmeter` label from it (if its name does not start with `upmeter-`), and the controller creates the project. Otherwise, move the workloads of the namespace to a namespace under another name. A namespace with an empty `projects.deckhouse.io/project` label is not in the list, `d8 k get namespaces -l 'projects.deckhouse.io/project='` lists such namespaces.

On adoption, a namespace that gets the `default` or `secure` template because of its monitoring or scanning label, and has no `security.deckhouse.io/pod-policy` label, gets `podSecurityProfile: ""`. The project puts no such label on it, and the default policy of the `admission-policy-engine` module keeps applying to it. A namespace whose label has a value other than `baseline`, `restricted` or `privileged` gets `podSecurityProfile: ""` too, and the label stays on it as it is. A namespace with none of these labels gets the `simple` template, which sets no profile and has no `podSecurityProfile` parameter.

To change the profile later, set `podSecurityProfile` in the project. For a project on `simple`, first move it to `default` with `networkPolicy: NotRestricted`, `requiredRequests: false` and `extendedMonitoringEnabled: false`, so that nothing else changes.

A namespace that already belongs to another Helm release (its `meta.helm.sh/release-name` annotation names a release other than the project's) is not taken over: the project gets the `HelmOwnership` condition with the name of the foreign release and retries, and the namespace is left as it is. Remove the foreign release or its ownership annotations to let the project proceed.

Existing RoleBinding and AuthorizationRule objects inside the namespace keep working after adoption. The namespace Admin is **not** copied into `.spec.administrators` and does **not** become `d8:project:admin`: that role additionally manages ProjectRoleBinding resources, which is a wider contract than in-namespace Admin. To make the team lead a project administrator, a platform operator adds them to `.spec.administrators` or creates a ProjectRoleBinding. The namespace Admin role allows only `get`, `list` and `watch` on the Namespace object.

The labels and annotations the module sets — `heritage`, `projects.deckhouse.io/*`,

`quota.projects.deckhouse.io/vm`, `quota.projects.deckhouse.io/lb`, `security.deckhouse.io/pod-policy`, `extended-monitoring.deckhouse.io/enabled`, `security-scanning.deckhouse.io/enabled`, `app.kubernetes.io/managed-by`, `meta.helm.sh/*`, the node-selector and tolerations annotations — are changed through the Project and its template; any other label or annotation (`istio-injection`, a Pod Security label, a GitOps tracking annotation) can still be set directly on the Namespace by anyone whose RBAC allows it, and the controller does not touch it. The `spec` and finalizers of the Namespace, and its deletion, stay with the controller.

This includes the labels and annotations the namespace had before it became a project. Adoption does not copy them into the project, so the project release does not own them, and the controller neither reverts a change to one of them nor brings one back once it is removed. They stay on the main namespace, and the additional namespaces of the project do not inherit them. Where such a label is needed on an additional namespace too, declare it in `spec.parameters.namespace.labels` of the Project or set it on that namespace directly.

The node-selector and tolerations annotations are the exception. They belong to the module, so adoption copies them into the `namespace.annotations` parameter of the project, where they are changed or removed. The pod security, monitoring and scanning labels are turned into parameters of the project, and the additional namespaces inherit them. The `extended-monitoring.deckhouse.io/enabled` and `security-scanning.deckhouse.io/enabled` labels keep the value `"true"` if the namespace had it. Any other value, `"false"` included, is replaced with an empty one, because adoption turns the feature on whenever the namespace carries the label, whatever its value.

The `projects.deckhouse.io/project` label itself is set only by the controller: a namespace created by hand with that label is refused, an empty value included, so a namespace cannot be made to look owned by a project it does not belong to. A namespace belongs to the project its label names and to no other: one without the label, or with an empty value, is not counted towards the project of the same name. A namespace that already carries the empty value is not adopted either: the empty value used to keep a namespace out of adoption, and it still does until the label is removed.

For example:

1. Create a new namespace:

   ```
   d8 k create ns test
   ```

2. Make sure that the project was created:

   ```
   d8 k get projects
   ```

   A new project corresponding to the namespace will appear in the project list:

   ```
   NAME        STATE      PROJECT TEMPLATE   DESCRIPTION
   AGE
   deckhouse   Deployed   virtual            This is a virtual project
   181d
   default     Deployed   virtual            This is a virtual project
   181d
   test        Deployed   simple
   1m
   ```

To change the template of such a project, set another template in `.spec.projectTemplateName`.

⚠ If the namespace already has an object of the same kind and name as one that the new template creates, for example the `isolated` NetworkPolicy, the project switches to `Error`, and the `ResourcesUpgraded` condition names the object. Delete or rename the object first.

Standard project fields

Project administrators and the resource quota are set in fields of the Project and work with any template (including `simple`):

- `.spec.administrators` — a list of subjects (`kind: User` or `kind: Group` and `name`) that receive administrative access to the project. The controller manages this access as an auto-generated ProjectRoleBinding in the project namespace. A name cannot contain control characters or line breaks and cannot start with `system:multitenancy-manager:`, the prefix of the users that

the controller acts as. An update that keeps such a name as it was is accepted with a warning.
- `.spec.quota` — a map of ResourceQuota hard limits (for example, `requests.cpu`, `limits.memory`) for the whole project. The controller turns it into the ProjectQuotaStatus of the project and reports the usage summed over the namespaces of the project in `.status.usage`; clearing the field deletes that ProjectQuotaStatus. Changing the field needs the right to set the quota of the project, and its keys follow the rules of the project quota. For `memory` and `storage`, a unit suffix is required (for example `2Gi`). Numbers without a unit mean bytes, and all of them except 0 are rejected. An update that leaves such a value as it was is accepted with a warning, so a project whose quota came from bare numbers in `parameters.resourceQuota` stays editable and can still be applied unchanged. A create through `deckhouse.io/v1alpha2` with such numbers in `parameters.resourceQuota` is accepted with a warning as well, so such a manifest can create the project again, for example from a backup.

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-project
spec:
  projectTemplateName: simple
  administrators:
    - kind: Group
      name: k8s-admins
  quota:
    requests.cpu: "5"
    requests.memory: 5Gi
    limits.cpu: "10"
    limits.memory: 10Gi
```

⚠ A Helm `resourcesTemplate` that renders a ResourceQuota or an AuthorizationRule keeps rendering it, next to the ResourceQuota and the ProjectRoleBinding that the standard fields create. A pod in such a project has to fit every ResourceQuota of its namespace, and the access such AuthorizationRules grant adds to the access of the standard fields.

Project quota

The project quota is one limit on the sum of resource usage over all namespaces of a project, the additional namespaces included. It is set in the `.spec.quota` field of the project:

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-project
spec:
  projectTemplateName: default
  quota:
    requests.cpu: "20"
    requests.memory: 40Gi
    projectNamespaces: "3"
```

From the field the controller makes the ProjectQuotaStatus object named after the project and keeps it up to date. The object shows the limits and the usage of the project and is read-only: only the controller creates, changes and deletes it. Clearing the field deletes it.

How the quota applies:

- The controller creates the `d8-project-quota` ResourceQuota with the limits of `spec.hard` in every namespace of the project, the ones added later included. This ResourceQuota counts the usage of its namespace, so no namespace alone uses more than the whole project may.
- A namespace that the quota applies in carries the `projects.deckhouse.io/quota: enforced` label once its `d8-project-quota` is calculated by kube-controller-manager. A namespace created for a ProjectNamespace carries `projects.deckhouse.io/quota: pending` from the start until then, and the main namespace of a project that has not been deployed yet gets it when the quota of the project is applied, so the manifests of the template wait for the quota too, however long the first attempts to deploy the project fail. The main namespace of a project that has been deployed

(in the `Deployed` state), as on an upgrade of the module or when a quota is set on a running project, is not held. The label is set by the controller only.
- In a namespace labelled `projects.deckhouse.io/quota: pending`, the `d8-multitenancy-manager-quota-pending` admission policy refuses the objects of users until `d8-project-quota` is calculated, and the ProjectNamespace reports `Ready=False`. That takes seconds after a namespace is created. In a namespace labelled `projects.deckhouse.io/quota: enforced`, the `d8-multitenancy-manager-quota-guard` admission policy refuses the objects the quota counts while `d8-project-quota` has no limits calculated at all or has scopes. A limit added to a quota that is calculated already does not hold anything back: it applies once kube-controller-manager has calculated it, and `Enforced` is `False` with the `KeyRollout` reason until then.
- A ProjectQuotaStatus is deleted in two steps: its `projects.deckhouse.io/project-quota` finalizer holds it until the controller has taken the label off the namespaces, and only then the `d8-project-quota` ResourceQuota objects go with it. A namespace that is being deleted loses the label at once, so that the objects waiting for a finalizer there, such as the volume claims protected from deletion, can be updated. If the module is disabled before the finalizer is removed, remove it by hand.

- The status of the ProjectQuotaStatus shows the usage summed over the namespaces (`status.used`), the number of namespaces and the 50 namespaces closest to a limit. The usage of every namespace is in its `d8-project-quota` ResourceQuota:

  ```
  d8 k get resourcequota -A -l projects.deckhouse.io/project=my-project
  ```

- In a namespace labelled `projects.deckhouse.io/quota: enforced`, creating an object that takes the usage of the whole project over a limit is denied with HTTP 403:

  ```
  admission webhook "quota-ledger.multitenancy-manager.deckhouse.io" denied the request: exceeded
  quota: d8-project-quota, project my-project: requested: requests.cpu=500m, used:
  requests.cpu=19800m, limited: requests.cpu=20 (3 namespaces)
  ```

  The webhook, called the quota ledger, keeps the sum of every project in the memory of its leader in the `quota-ledger` container of the module pod. While no replica answers (for a few seconds when the leader changes), a counted create in a namespace of a quota project fails with `project quota ledger unavailable (<CAUSE>), retry`; deletions, running objects and namespaces without the quota are not affected. A Deployment or a Job retries such a create by itself. The same answer comes at once, with the `too many approvals of the project are not confirmed yet` cause, for a write that would wait for the earlier approvals of its project to be confirmed while two writes from the same API server already wait: the third waiting write of a project from one API server gets a 503, and its controller retries it.

- The sum is not checked for a request with `--dry-run=server`: the API server does not write the usage of a dry run. The quota of the namespace is still checked.
- The `Enforced` condition says whether the quota applies in every namespace, and the `Exceeded` condition whether the usage is above a limit. A limit may be lowered below the current usage: the change is accepted with a warning, nothing running is evicted, and `Exceeded` is `True` with the `HardLowered` reason. The `D8ProjectQuotaExceeded` alert fires then.

Keys of a project quota

`.spec.quota` accepts the following keys:

- `cpu`, `memory`, `ephemeral-storage`, `pods`, `requests.*` and `limits.*` of CPU, memory and ephemeral storage, `requests.storage`, `requests.hugepages-<SIZE>`, the requests of extended resources (for example, `requests.nvidia.com/gpu`);
- `services`, `services.loadbalancers`, `services.nodeports`, `persistentvolumeclaims`, `configmaps`, `secrets`, `replicationcontrollers`;
- `<CLASS>.storageclass.storage.k8s.io/requests.storage` and `<CLASS>.storageclass.storage.k8s.io/persistentvolumeclaims`, `<CLASS>.deviceclass.resource.k8s.io/devices`;
- `count/<RESOURCE>[.<GROUP>]` for a namespaced resource the API server serves, for example `count/ingresses.networking.k8s.io`;
- `projectNamespaces`, an alias of `count/projectnamespaces.deckhouse.io`: the number of ProjectNamespace objects of the project, that is, of its additional namespaces. The alias is expanded in the ResourceQuota objects, and the object keeps it as written.
- the keys of virtual machines and of load balancers, which no ResourceQuota counts.

Refused are `resourcequotas` and `count/resourcequotas` (`d8-project-quota` takes one in every namespace by itself), the count of a cluster-scoped resource such as `count/namespaces`, a resource misspelled in a served group, a `count/` key of a resource that lacks one of the verbs create, list, watch and delete, and the keys with the reserved `projectquota.deckhouse.io/` prefix. A `count/` key of a group the API server does not serve yet is accepted with a warning: nothing counts it until the module

that provides the group is enabled, and `Enforced` is `False` with the `KeyNotServed` reason until then. A change that keeps a key the object already has is accepted with a warning even when the key would be refused now.

A new key in a live quota starts to apply once `d8-project-quota` of every namespace has it calculated; until then `Enforced` is `False` with the `KeyRollout` reason.

Pods and volume claims with `resource-quota-overrides.deckhouse.io/*` labels set by platform controllers are counted as a ResourceQuota counts them.

Virtual machines in a project quota

Where the `virtualization` module is enabled, a project quota limits the virtual machines of the project by their specification. No ResourceQuota counts these keys, so they do not go into `d8-project-quota`: the quota ledger checks every create of a VirtualMachine and every change of its `spec` in the namespaces of a project whose quota has them, and counts the virtual machines itself in these keys:

- `virtualmachines.virtualization.deckhouse.io/cores`: Sum of `spec.cpu.cores`.
- `virtualmachines.virtualization.deckhouse.io/cpu`: Guaranteed CPU, the sum of cores × `spec.cpu.coreFraction`. An empty `spec.cpu.coreFraction` or `Auto` counts as whole cores.
- `virtualmachines.virtualization.deckhouse.io/memory`: Sum of `spec.memory.size`.
- `virtualmachines.virtualization.deckhouse.io/gpus`: Number of GPUs in `spec.gpus`.
- `<GPU CLASS>.gpuclass.virtualization.deckhouse.io/gpus`: Number of GPUs of the `<GPU_CLASS>` GPUClass.

The number of virtual machines is limited by the ordinary `count/virtualmachines.virtualization.deckhouse.io` key.

Every virtual machine counts in the keys above, a stopped one too: a machine stopped by hand may be started at any moment. Only a machine with `runPolicy: AlwaysOff`, which is kept switched off on purpose, is not counted in them; the `count/virtualmachines.virtualization.deckhouse.io` key counts such machines too. A change that does not raise a key, such as making a machine smaller or stopping it, is never refused over the quota and passes even while the quota ledger does not answer; every other create and change of `spec` needs the ledger to answer. A change that touches only the metadata of a machine does not call the ledger. A smaller spec frees the quota only once the machine runs with it: until a restart applies it, the machine counts with the cores and memory it runs with (`status.resources`) and with the GPUs that the changes in `status.restartAwaitingChanges` show. Until the `virtualization` module has processed a change of the spec, the machine keeps what it counted before. The namespaces of such a project carry the `quota.projects.deckhouse.io/vm: enforced` label, which the controller sets. While the `virtualization` module is disabled, the keys count nothing, and `Enforced` is `False` with the `KeyNotServed` reason. The quota is accepted with a warning when the API server does not serve VirtualMachine.

The pod of a virtual machine counts in `requests.*` and `limits.*` as well, twice while the machine migrates. A quota that has both the keys of virtual machines and the keys of pods (`cpu`, `memory`, `requests.*`, `limits.*`) is accepted with a warning. Limit virtual machines with their keys and containers with `requests.*` and `limits.*`.

Load balancers in a project quota

A project quota limits the Services of type LoadBalancer of the project by their class, by their pool and by the external addresses they take. No ResourceQuota counts these keys, so they do not go into `d8-project-quota`: the quota ledger checks every create of a Service that is or was of type LoadBalancer, and every change of its `spec` or of the `network.deckhouse.io/metal-load-balancer-class`, `network.deckhouse.io/l2-load-balancer-external-ips-count`, `network.deckhouse.io/load-balancer-ips` and `network.deckhouse.io/load-balancer-pool` annotations, in the namespaces of a project whose quota has them, and counts the Services itself in these keys:

- `<CLASS>.loadbalancerclass.network.deckhouse.io/services`: Number of load balancers of the `<CLASS>` class.
- `<CLASS>.loadbalancerclass.network.deckhouse.io/addresses`: External addresses of the `<CLASS>` class.
- `<POOL>.loadbalancerpool.network.deckhouse.io/addresses`: Addresses taken from the `<POOL>` pool of the BGP mode.
- `loadbalancers.network.deckhouse.io/addresses`: All external addresses of the project.

The number of load balancers of any class is limited by the ordinary `services.loadbalancers` key.

The class of a Service is the one the `metallb` module serves it with. It comes from the first source that applies:

- The class the `metallb` module has bound the Service to in its status, even if no such MetalLoadBalancerClass exists

- `spec.loadBalancerClass`, when such a MetalLoadBalancerClass exists
- The `network.deckhouse.io/metal-load-balancer-class` annotation, when such a MetalLoadBalancerClass exists
- The MetalLoadBalancerClass with `isDefault: true`
- `spec.loadBalancerClass` without a check, the class of another load balancer controller
- `default`, the load balancers of a cloud provider

Where the API server does not serve MetalLoadBalancerClass, no class is checked: the class is `spec.loadBalancerClass`, else `default`, and the annotation, which only the `metallb` module reads, does not count. The quota ledger notices within 30 seconds that the `metallb` module has been enabled or disabled, and counts the load balancers again without a restart.

A Service with the `network.deckhouse.io/load-balancer-pool` annotation takes one address of that pool. Any other Service takes the number of addresses from its `network.deckhouse.io/l2-load-balancer-external-ips-count` annotation when the value is a number above one, and one address when the annotation has any other value. The `network.deckhouse.io/load-balancer-ips` annotation only picks which addresses the Service takes, as the `metallb` module assigns them.

Every Service of type LoadBalancer counts its own addresses, also when it shares an address with other Services under the `network.deckhouse.io/load-balancer-shared-ip-key` annotation: two Services that share one address take two addresses of the quota. The namespaces of such a project carry the `quota.projects.deckhouse.io/lb: enforced` label, which the controller sets.

Clusters where Deckhouse Platform does not manage the control plane

The sum over the namespaces of a project is enforced only where Deckhouse Platform manages the control plane of the cluster (the `control-plane-manager` module is enabled). Elsewhere the `d8-multitenancy-manager-quota-pending` and `d8-multitenancy-manager-quota-guard` admission policies are not rendered, every ProjectQuotaStatus reports `Enforced=False` with the `ControlPlaneNotManaged` reason, and the `D8ProjectQuotaNotSupported` alert says so. Each namespace still keeps its `d8-project-quota` ResourceQuota, so no namespace alone uses more than the quota of the whole project.

Dividing a project quota between namespaces

A project administrator may create ResourceQuota objects of their own in the namespaces of the project. Such a quota only narrows the project quota: an object has to fit every ResourceQuota of its namespace. A quota of a project administrator cannot be named `d8-project-quota` and cannot limit `resourcequotas` or `count/resourcequotas`. A `d8-project-quota` that the controller did not create, or one with scopes, is deleted and created again by the controller.

For example, to keep the `my-project-db` namespace within a quarter of the CPU of the project:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: db-share
  namespace: my-project-db
spec:
  hard:
    requests.cpu: "5"
```

Who manages the quotas

Changing `.spec.quota` of a Project needs the `update` verb on `projects/quota` in the `deckhouse.io` group for that project, which the `d8:system-capability:multitenancy-manager:manage_quotas` capability grants; the IAM `manager` role and the roles above it include the capability. The Project webhook checks the right, so whoever may create and change projects does not choose their quota. A ProjectQuotaStatus is written by the controller only; everyone else, cluster administrators included, can only read it. The usage in the status of `d8-project-quota` is written by Kubernetes; it is protected from other writers where Deckhouse Platform manages the control plane.

When the project quota refuses

What the response says:

Response	What it means
`403 ... exceeded quota: d8-project-quota, project <PROJECT NAME>: requested: ..., used: ..., limited: ... (<N> namespaces)`	The sum of project `<PROJECT_NAME>` would go over the limits of its quota. Raise the limits in `.spec.quota` of the project or delete what the project does not need; `status.topNamespaces` of its ProjectQuotaStatus shows where the usage is.
`403 ... exceeded quota: d8-project-quota, requested: ...` (no `project`)	The usage of this namespace alone, with the object, would go over a limit of the project, which every `d8-project-quota` ResourceQuota contains. The quota plugin of the API server refuses it before the sum of the project is checked.
`503 ... project quota ledger unavailable (<CAUSE>), retry`	The ledger does not answer for a moment, for example while its leader changes. Retry; if it lasts longer than a minute, see the `D8ProjectQuotaLedgerDown` alert.
`500 ... failed calling webhook "quota-ledger.multitenancy-manager.deckhouse.io": ...`, or the same with `quota-vm.multitenancy-manager.deckhouse.io` or `quota-lb.multitenancy-manager.deckhouse.io`	The API server does not reach the ledger: no replica runs, the network, or the certificate. While the Service points at a frozen leader, the quota plugin of the API server retries the call, so each such create waits up to about 8 seconds until the address of the leader is taken out of the Service: in about 3.5 seconds by the other replica, in up to about 15 seconds by Deckhouse when no replica answers.
`403 ... ValidatingAdmissionPolicy 'd8-multitenancy-manager-quota-pending' ...`	The `d8-project-quota` of a new namespace is not calculated yet. Retry in a few seconds; if it lasts, see the `Enforced` condition of the ProjectQuotaStatus of the project and check the controller and kube-controller-manager.
`403 ... ValidatingAdmissionPolicy 'd8-multitenancy-manager-quota-guard' ...`	The `d8-project-quota` of the namespace is missing, has scopes or is not calculated at all yet. Retry in a few seconds; otherwise check the controller and kube-controller-manager.
`500 ... resource quota evaluation timed out`	The quota admission of the API server is overloaded; see the `D8ResourceQuotaAdmissionDegraded` alert.

A new additional namespace refuses the objects of users for the few seconds until its `d8-project-quota` is calculated. A namespace where the quota applies and whose `d8-project-quota` is gone does the same until the controller makes it anew, and refuses the objects of `system:sudouser` too. Of the users, only `system:sudouser` can delete `d8-project-quota`.

If the ledger keeps the objects of projects from being created and the cause cannot be removed quickly, take these steps in order until the creates work again:

1. If the creates fail with `503 ... project quota ledger unavailable`, a replica answers, and suspending the ledger helps. Within a second every running replica lets the writes through uncounted, and every ProjectQuotaStatus reports `Enforced=False` with the `Suspended` reason. Deleting the ConfigMap resumes the check; the `D8ProjectQuotaSuspended` alert reminds of it after 15 minutes.

   ```
   d8 k -n d8-multitenancy-manager create configmap quota-ledger-suspend --from-literal=reason="
   <REASON>"
   d8 k -n d8-multitenancy-manager delete configmap quota-ledger-suspend
   ```

   Replace `<REASON>` with why the check is suspended.

2. If the creates fail with `500 ... failed calling webhook "quota-ledger.multitenancy-manager.deckhouse.io"`, the API servers do not reach a replica that answers, as while the `D8ProjectQuotaLedgerDown` alert fires, and the suspension does not help: delete the webhook of the ledger as `system:sudouser`. The check stops within a second, and every ProjectQuotaStatus reports `Enforced=False` with the `WebhookMissing` reason. Deckhouse Platform finds the webhook of the module missing and puts it back by itself within about 4 minutes, so this buys time to fix the cause rather than switching the check off.

```
d8 k --as=system:sudouser delete validatingwebhookconfiguration d8-multitenancy-manager-quota-ledger
```

Check that the webhook is back: `d8 k get validatingwebhookconfiguration d8-multitenancy-manager-quota-ledger`.

3. Disable the `multitenancy-manager` module. The check goes away together with everything else the module does for projects.

Before draining the worker nodes of a cluster with project quotas (evicting the pods from the nodes with `d8 k drain` or on a node update), check that the ledger has a leader (`d8_multitenancy_quota_ledger_leader`) and that the `D8ProjectQuotaLedgerCanaryFailing` alert is not firing: while the ledger does not answer, the pods that replace the evicted ones are not created in the namespaces of projects with a quota.

Granting access within a project

To grant access to project namespaces for users beyond the project administrators, use role bindings that reference cluster-wide roles and fan out into the appropriate project namespaces automatically:

- ProjectRoleBinding (namespaced, short name `prb`) — grants a role within a **single** project. It must be created in the project's main namespace (the namespace whose name equals the project name). The controller creates a RoleBinding in every namespace of that project.
- ClusterProjectRoleBinding (cluster-scoped, short name `cprb`) — grants a role across **all** non-virtual projects. The controller creates a RoleBinding in every namespace of every project and reports the number of bound projects in `.status.boundProjects`.

`roleRef` must reference a ClusterRole whose name starts with one of the allowed prefixes (`d8:project:`, `d8:namespace:`, `d8:project-capability:`, `d8:namespace-capability:`, `d8:custom:`). The roles are described in the `user-authz` module documentation.

The following checks apply when bindings are created:

- **Privilege escalation protection**: a binding can only be created by a user who has the right to bind (`bind`) the referenced role, checked by name. A project administrator (`d8:project:admin`) holds `bind` on exactly eight roles: `d8:project:viewer`, `d8:project:user`, `d8:project:manager`, `d8:project:admin`, `d8:namespace:viewer`, `d8:namespace:user`, `d8:namespace:manager` and `d8:namespace:admin`. A binding to any other role — a custom `d8:custom:*` role, a capability, `d8:project:superadmin` — is refused for the project administrator even when the role is narrower than their own permissions, because nothing grants them `bind` on that name. Such bindings are created by a cluster administrator, or the cluster administrator grants `bind` on the custom role to the project administrators with a ClusterRole of their own.
- The role must exist: a binding to a non-existent role is rejected.
- A ServiceAccount used as a subject of a ProjectRoleBinding must belong to a namespace of that same project. Its main namespace, its additional namespaces and a namespace a ProjectNamespace of the project has claimed qualify. A namespace that is only named like an additional namespace (`<project>-*`) but belongs to a separate project does not. When the binding is updated, a ServiceAccount the stored binding already has is not refused again. If its namespace is no longer a namespace of the project, the response carries a warning that this ServiceAccount gets no rights from the binding.
- System and subsystem roles (`d8:system:*`, `d8:subsystem:*`), as well as arbitrary roles outside the listed prefixes, cannot be granted via project bindings.
- Roles with the `rbac.deckhouse.io/disabled-for-direct-use-in-projects: "true"` annotation are forbidden for granting in projects. A cluster administrator can put this annotation on a role to phase it out: existing bindings keep working, but new ones cannot be created. If such a role is used by a project template, the project switches to the `Error` state with an explanation in the `TemplateRolesAllowed` condition.

The controller checks the ServiceAccounts by the same rule every time it updates the RoleBindings of a ProjectRoleBinding, and also when a namespace appears, is deleted or moves to another project. A subject that names ServiceAccounts goes into the RoleBindings only while their namespace is the main namespace of the project, or exists, is not being deleted and carries the `projects.deckhouse.io/project` label with the project name. Besides subjects of the `ServiceAccount` kind, this applies to the `User` subject named `system:serviceaccount:<namespace>:<name>` and the `Group` subject named `system:serviceaccounts:<namespace>`, which RBAC matches to the same ServiceAccounts. Other users and groups are not affected. So a ServiceAccount of a deleted additional namespace gets no rights when a separate project takes the name later.

The left-out subjects stay in `.spec.subjects`. The `SubjectsBound` condition of the binding is `False` and lists them, and the `SubjectsLeftOut` Warning event is sent when the list changes. The `Ready`

condition stays `True`, as the binding keeps working for the other subjects. Once the namespace is a namespace of the project again, for example after the deleted ProjectNamespace is created again, the subjects get the rights back without a change to the binding. You can check the condition in the status of the binding:

```
d8 k -n my-project get projectrolebinding ci -o jsonpath='{.status.conditions[?
(@.type=="SubjectsBound")]}'
```

The `d8-administrators` binding created by the controller from the `.spec.administrators` field is managed by the controller only — it cannot be edited manually. To change the set of administrators, change the `.spec.administrators` field of the project.

The ServiceAccount rule above does not apply to this binding. An administrator given as the `User system:serviceaccount:<namespace>:<name>` or the `Group system:serviceaccounts:<namespace>` gets the rights in every namespace of the project, whichever project that namespace belongs to, because only those allowed to change the Project set its administrators. Remove such an administrator from the project before its namespace is deleted, as the ServiceAccounts of a namespace created later under the same name would get these rights.

The following bindings give the `viewer@example.com` user the `d8:project:viewer` role in every namespace of `my-project`, and the `platform` group the same role in every project except the virtual ones:

```
---
apiVersion: deckhouse.io/v1alpha3
kind: ProjectRoleBinding
metadata:
  name: viewers
  namespace: my-project
spec:
  subjects:
    - kind: User
      name: viewer@example.com
  roleRef:
    kind: ClusterRole
    name: d8:project:viewer
---
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: platform-viewers
spec:
  subjects:
    - kind: Group
      name: platform
  roleRef:
    kind: ClusterRole
    name: d8:project:viewer
```

Roles available in a RoleBinding inside a project

Besides the project bindings, a plain RoleBinding can also be used inside a project namespace — the role then applies in that single namespace only. However, in projects the set of roles available to a plain RoleBinding is restricted: only cluster roles carrying the `rbac.deckhouse.io/delegatable: "true"` label are allowed. Among the built-in ones these are the `d8:namespace:*` and `d8:project:*` roles, as well as the access-level roles of the basic role model (`user-authz:user`, `user-authz:privileged-user`, `user-authz:editor`, `user-authz:admin`).

A RoleBinding to any other cluster role (for example, `cluster-admin`, system roles, or capabilities) is rejected in a project with the message `references "<role>" which is not available to project`. This protects the project isolation from being bypassed by binding to an overly broad role.

To use a custom role in projects, add the `rbac.deckhouse.io/delegatable: "true"` label to it:

```
d8 k label clusterrole d8:custom:namespace:developer rbac.deckhouse.io/delegatable=true
```

The restriction applies in the namespaces of every project, including the ones adopted from a standalone namespace.

Structured templates

A `deckhouse.io/v1alpha2` project template is described by **structured fields**. You declaratively specify which settings the project namespaces get. The controller itself creates the corresponding objects (network policies, security policies, log collection settings, etc.) from these fields in every namespace of the project and keeps them up to date. Other objects of the project namespaces are described in `manifests`.

Available fields (all optional; the complete reference is in the ProjectTemplate resource description):

Field	What it configures
`podSecurityStandard`	Pod security profile: `Privileged`, `Baseline`, or `Restricted`. A parameter with an empty value sets no profile
`networkPolicy.mode`	Network isolation: `Isolated` (traffic is only allowed within the project and from the platform system components) or `NotRestricted`
`features.monitoring`	Extended monitoring of the project namespaces
`features.vulnerabilityScanning`	Scanning of container images for vulnerabilities
`logShipping.clusterDestinationRef`	Collecting the logs of the project pods into the given destination (`ClusterLogDestination`)
`nodeSelector`, `tolerations`	Placing the pods of the main namespace on dedicated nodes
`allowedUIDs`, `allowedGIDs`	The allowed UID/GID ranges of the project containers
`runtimeAudit.enabled`	Auditing the project processes’ access to the Linux kernel
`namespaceMetadata.labels`, `namespaceMetadata.annotations`	Extra labels and annotations of the project namespaces. The labels the module manages cannot be set here. A template or a Project that sets one is refused. One that already has such a label can still be changed while it keeps the label as it is. The label is not applied, and the response warns about it
`resources`, `grantPolicies`	Granting cluster-wide resources through a project template
`manifests`	Other objects in the project namespaces, applied with the rights of a project administrator. For details, see Describing other objects in manifests
`parametersSchema.openAPIV3Schema`	The schema of parameters set when creating a project

An example of a structured template:

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: my-template
spec:
  title: "Team template"
  description: "Isolated project with monitoring"
  podSecurityStandard: Baseline
  networkPolicy:
    mode: Isolated
  features:
    monitoring: true
    vulnerabilityScanning: true
```

Template parametrization

The following fields of the template can take a parameter: `podSecurityStandard`, `networkPolicy.mode`, `features.monitoring`, `features.vulnerabilityScanning`, `logShipping.clusterDestinationRef`, `nodeSelector`, `tolerations`, `allowedUIDs`, `allowedGIDs`, `runtimeAudit.enabled`, `namespaceMetadata.labels` and `namespaceMetadata.annotations`. Instead of a concrete value, specify `{fromParam: <parameter name>}` and declare the parameter in `parametersSchema`. The value does not have to be a scalar: a map (`nodeSelector`, `namespaceMetadata.labels`), a list (`tolerations`) or an object (`allowedUIDs`) works just as well. The other fields — `title`, `description`, `resources`, `grantPolicies` and `parametersSchema` itself — take literal values only. Each project then sets its own value in `.spec.parameters`; if the value is not set, the `default` from the schema is used.

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: my-parametrized-template
spec:
  podSecurityStandard:
    fromParam: securityProfile
  networkPolicy:
    mode:
      fromParam: networkMode
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        securityProfile:
          type: string
          enum: [Baseline, Restricted]
          default: Baseline
        networkMode:
          type: string
          enum: [Isolated, NotRestricted]
          default: Isolated
```

A project using such a template:

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-project
spec:
  projectTemplateName: my-parametrized-template
  parameters:
    securityProfile: Restricted
```

A template whose `parametersSchema` still declares the `administrators` and `resourceQuota` parameters keeps accepting projects. A schema declares them by name, or through an `additionalProperties` that takes every parameter it does not name, as long as it takes these two in the form a `v1alpha2` Project holds them. The project parameters are validated against such a schema together with `.spec.administrators` and `.spec.quota`, laid out again as `parameters.administrators` and `parameters.resourceQuota` the way the conversion webhook lays them out for `v1alpha2`. A quota value is checked in the first form the schema takes among its canonical spelling (for example `1Gi` for `1024Mi`), a number (for example `1000` for `1k`) and a spelling with another unit (for example `2000m` for `2` when the schema has `pattern: ^[0-9]+m$`), in that order. A required field that the project leaves empty is checked in its empty form (`[]` for the administrators, `{}` for the quota). A Helm `resourcesTemplate` and `manifests` receive the parameters laid out the same way in `.parameters`, so a text written for `v1alpha2` projects keeps rendering its objects from them. A quota value reaches the text in the form it is checked in, so for a schema that takes strings a zero is the string `"0"`, which `with` and `if` take as set. The structured fields do not receive these values as parameters.

A schema that neither names the `administrators` and `resourceQuota` parameters nor takes unnamed parameters through `additionalProperties` does not declare them, and the schemas of the built-in templates do not declare them either. On a template with such a schema, these parameters

grant no access and set no quota. A project that holds them, for example an empty list or a quota with the number `0` that the conversion webhook leaves in `parameters`, keeps working and can be edited. An update that keeps such a value as it was, including on a move to another template that does not declare them either, is accepted with a warning, and so are an empty list and a quota with no value other than the number `0`. Any other new or changed value is rejected. Set the administrators in `.spec.administrators` and the quota in `.spec.quota` instead. In a `v1alpha2` manifest, write them in a form that the conversion webhook lifts into these fields. Any other parameter that the schema does not declare is still rejected.

The `fromParam` references are checked when a template is created or changed: a reference to an undeclared parameter or to a parameter of an incompatible type (for example, a string parameter for a boolean field) is rejected.

A declared parameter is type-checked even where the schema allows undeclared keys with `additionalProperties` or `x-kubernetes-preserve-unknown-fields`. A reference to an undeclared key is accepted only in such places, and the type of such a key is not checked.

The `requiredRequests` parameter is read by name, without `fromParam`. When the template declares it in `parametersSchema` with `type: boolean` and its value for the project is `true`, the controller renders an OperationPolicy in the `Deny` mode that requires CPU and memory requests from the pods in every namespace of the project. A template that does not declare the parameter renders no such policy, even if its schema allows `additionalProperties` and the project sets `requiredRequests` as an undeclared key. To keep the policy on unless a project turns it off, declare the parameter with `default: true`, as the `default`, `secure` and `secure-with-dedicated-nodes` templates do:

```
spec:
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        requiredRequests:
          type: boolean
          default: true
```

Template checks

The following rules apply to template operations:

- A template cannot be deleted while at least one project uses it or a project namespace is still rendered from it. After a project switches to another template, the old template stays until the new one has rendered.
- A change to a template is automatically applied to all projects created from it.
- Instead of structured fields, a template can describe its objects with a Helm text in `resourcesTemplate`. Such a template is rendered through Helm with the project name (`.projectName`), the template name (`.projectTemplateName`) and the project parameters (`.parameters`), and `resourcesTemplate` cannot be set together with the fields that render namespace objects or with `manifests`. A template that projects use keeps a non-empty `resourcesTemplate` until a change sets such fields or `manifests` in its place. What such a change, or a move of a project to another template, does with the objects the Helm text rendered is described in Switching a project from a Helm text. A project whose parameters turn into objects or keys of the rendered manifests instead of values is refused, so quote every substitution. This form is deprecated, and `resourcesTemplate` goes away from `deckhouse.io/v1alpha2` together with `deckhouse.io/v1alpha1` in a future release. The `MultitenancyManagerHelmProjectTemplate` alert names every template that still renders from a Helm text, except the built-in ones. Describe new templates with structured fields and `manifests`, and move the existing ones as Moving a template from a Helm text describes.
- A template cannot get a new `resourcesTemplate` through either version. A create with a non-empty `resourcesTemplate` and an update that sets it on a template whose `resourcesTemplate` is empty are refused. A template that has a Helm text keeps it, and the text can be changed through either version while `deckhouse.io/v1alpha1` can describe the whole template. So a template with a Helm text cannot be created again, for example from a manifest exported from another cluster, from a backup, or from Git after the template was deleted. Move such a template to structured fields and `manifests` first.
- The `deckhouse.io/v1alpha1` version of ProjectTemplate is served and deprecated, and the API server returns a warning on every request through it. A template is created through `deckhouse.io/v1alpha2` only, and a create through `deckhouse.io/v1alpha1` is refused. The templates created through `deckhouse.io/v1alpha1` earlier are read, changed and deleted through it as before. This version has the `description`, `parametersSchema` and `resourcesTemplate` fields, and `deckhouse.io/v1alpha2` keeps all three, so a template written as `v1alpha1` reads back unchanged. A change through `v1alpha1` to a template that has fields

`v1alpha1` cannot describe, such as `title`, `grantPolicies`, `podSecurityStandard` or `manifests`, is refused, because it would erase them. Change such a template through `deckhouse.io/v1alpha2`.

Describing other objects in manifests

The structured fields cover the settings that most projects need. Other objects of the project namespaces, such as workloads, ConfigMap, RoleBinding or LimitRange objects, are described in `manifests`. It is a text in the `go template` format with the `helm` functions, like `resourcesTemplate`, and a template can set it together with the structured fields.

`manifests` and `resourcesTemplate` differ in whose rights create the objects. The controller installs a Helm text from `resourcesTemplate` as a Helm release with its own rights, which cover the whole cluster. So such a text can create any cluster-scoped object other than a namespace, and any object in the namespaces of the project. The controller applies the objects of `manifests` as a user of the project that holds the `d8:project:admin` role in the namespaces of the project. A template thus gets the rights of an administrator of the project, not the rights of the controller.

The text gets the project name in `.projectName`, the template name in `.projectTemplateName` and the project parameters completed with the defaults of `parametersSchema` in `.parameters`, the same values that `resourcesTemplate` gets. A project whose parameters turn into objects or keys of the rendered manifests instead of values is refused, so quote every substitution. The text is rendered without access to the cluster, so the `lookup` function finds nothing. A project template can be read by users who have no access to its projects, the ones with the `User` access level included, so do not put secret values into `manifests`.

The objects are applied as follows:

- The controller creates the `d8-manifests` ProjectRoleBinding in the main namespace of the project. It grants the `d8:project:admin` role to the `system:multitenancy-manager:project:<PROJECT_NAME>` user in every namespace of the project.
- The objects are applied once this binding is `Ready` for its current generation and the `d8:project:admin` ClusterRole has its aggregated rules. Until then the `ManifestsApplied` condition of the project says what the objects wait for, and the project stays `Deployed`.
- Each object is applied with server-side apply as this user, with the `multitenancy-manager/manifests` field manager. An object without `metadata.namespace` goes to the main namespace of the project. An object can also go to an additional namespace of the project, including one that a ProjectNamespace of the same `manifests` makes, as Additional namespaces in manifests describes.
- Each object gets the `projects.deckhouse.io/project: <PROJECT NAME>` and `projects.deckhouse.io/source: manifests` labels, and `.status.manifestObjects` of the project lists the objects.
- An object that exists already is applied only when it belongs to the project, that is, when `.status.manifestObjects` lists it, when a switch from a Helm text left it in `.status.keptObjects`, or when it has both labels. Any other object is left alone, as Limitations of manifests describes.
- The objects do not get the `heritage: multitenancy-manager` label. They get the `projects.deckhouse.io/source: manifests` label instead, and like the objects of the structured fields and of a Helm text, they are changed through the template only. The `d8-multitenancy-manager-manifests-objects` admission policy refuses to create, change or delete such an object to everyone, project and cluster administrators included. Only the user the `manifests` of the project are applied as, the controller, the service accounts of the `d8-*` namespaces and the cluster components can do it. Anyone who may update a workload may restart its rollout, when the `kubectl.kubernetes.io/restartedAt` annotation of its pod template is the only change. `metadata.resourceVersion`, `metadata.generation` and `metadata.managedFields` are not compared, because the API server sets them on the write. Within 10 minutes the controller applies the objects again, which puts back the fields that the template sets if something else changed them.
- An object that the template no longer renders is deleted as the same user after the other objects are applied, and only when none of them was refused or left alone. It is deleted through the version that the API server serves its kind at, also when it was applied through another one. An object that loses one of the two labels, and an object whose namespace left the project, leave the list and stay in place. Only the service accounts of the `d8-*` namespaces, the controller among them, and the cluster components can take a label off. The user the `manifests` are applied as can take off only the `projects.deckhouse.io/source` label.
- When one change makes the project release render an object that `manifests` applied, under the same kind, namespace and name, the release takes the object over, and the object stays in the cluster. Until the upgrade finishes, `.status.handingOver` lists the object and `.status.manifestObjects` does not. If the upgrade does not deploy the object, it goes back to `manifests`, and `.status.manifestObjects` lists it again.
- When the same change makes the release render another object that exists without the ownership marks of the release and is not an object of `manifests`, Helm cannot take that object

over. The ownership marks are the `app.kubernetes.io/managed-by: Helm` label and the `meta.helm.sh/release-name` annotation with the name of the project release. The controller does not run the upgrade then and leaves the objects of `manifests` unchanged, and the `ResourcesUpgraded` condition of the project names the object that blocks the upgrade. The handover goes on once that object is deleted or gets the ownership marks.
- When the template drops `manifests` or the project moves to a template without it, the objects are deleted, and then the `d8-manifests` binding is deleted. When the project is deleted, the binding is deleted first, and the objects go with the namespaces.

Only the controller binds the users whose names start with `system:multitenancy-manager:`. A RoleBinding, ClusterRoleBinding, ProjectRoleBinding, ClusterProjectRoleBinding, AuthorizationRule or ClusterAuthorizationRule that names such a user or group is refused for everyone else, except the garbage collector and `system:sudouser` (see Who bypasses admission). A Helm text in `resourcesTemplate` can also bind such a user, because the controller installs it with its own rights. Apart from that, the objects of a template get the rights of the `d8-manifests` binding and the rights that the cluster grants to every authenticated user (the `system:authenticated` group). For the same reason a project cannot name such a user or group among its administrators, in `.spec.administrators` or in the `administrators` parameter. A name that the project already has is accepted with a warning, so the project stays editable and the name can be removed.

The following template sets the pod security profile and the network isolation with structured fields and creates two objects in `manifests`, a LimitRange with the default limits from the project parameters and a ConfigMap with the names of the project and of its template:

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: team-with-limits
spec:
  podSecurityStandard: Baseline
  networkPolicy:
    mode: Isolated
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        defaultCPU:
          type: string
          default: 500m
        defaultMemory:
          type: string
          default: 512Mi
  manifests: |
    apiVersion: v1
    kind: LimitRange
    metadata:
      name: container-defaults
    spec:
      limits:
        - type: Container
          default:
            cpu: {{ .parameters.defaultCPU | quote }}
            memory: {{ .parameters.defaultMemory | quote }}
    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: project-info
    data:
      project: {{ .projectName | quote }}
      template: {{ .projectTemplateName | quote }}
```

Once the `ManifestsApplied` condition is `True`, the main namespace of every project of the template has the `container-defaults` LimitRange and the `project-info` ConfigMap. To check the condition and to list the applied objects, use the commands:

```
d8 k get project <PROJECT_NAME> -o jsonpath='{.status.conditions[?
(@.type=="ManifestsApplied")]}'
d8 k get project <PROJECT_NAME> -o jsonpath='{range .status.manifestObjects[*]}{.kind}
{.namespace}/{.name}{"\n"}{end}'
```

Additional namespaces in manifests

`manifests` can make additional namespaces of the project with ProjectNamespace objects and put objects into them in the same text. A ProjectNamespace without `metadata.namespace` goes to the main namespace of the project, and its namespace is named `<PROJECT_NAME>-<spec.name>`, so an object in it names the namespace as `{{ .projectName }}-<spec.name>`.

The namespace does not exist on the first render, so the objects are applied in two steps:

- The ProjectNamespace objects and the objects of the namespaces that exist are applied at once.
- The objects of a namespace that a ProjectNamespace of the same `manifests` makes wait until the namespace exists and the `d8-manifests` binding has reached it. Until then the `ManifestsApplied` condition is `False` and names what the objects wait for, and the project stays `Deployed`. A namespace that is being deleted is reported as terminating, and one that is not there yet as still to be created. When the ProjectNamespace is not `Ready`, the condition includes its `Ready=False` message. Objects that wait are kept. Objects that `manifests` no longer renders are deleted. The project is reconciled when the namespace is created and when the binding reaches it, so the objects are applied without any action.
- When `manifests` drops a ProjectNamespace, the ProjectNamespace is deleted, and its namespace goes with everything in it. The objects of `manifests` in that namespace leave `.status.manifestObjects` without a delete of their own.

An object in a namespace that is neither a namespace of the project nor made by a ProjectNamespace of the same `manifests` is refused, as Limitations of manifests describes. A create or a change of a project whose `manifests` render such an object is admitted with a warning that names the namespace.

The following `manifests` make the `backend` namespace and put a ConfigMap into it:

```
manifests: |
  apiVersion: deckhouse.io/v1alpha3
  kind: ProjectNamespace
  metadata:
    name: backend
  spec:
    name: backend
  ---
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: backend-settings
    namespace: "{{ .projectName }}-backend"
  data:
    team: {{ .parameters.team | quote }}
```

Limitations of manifests

`manifests` covers only what an administrator of the project can do in its namespaces. The following limitations apply:

- A text that does not render for a project, for example because of a syntax error or a `required` value that the project does not set, is accepted when the template is written, because the template is not rendered at that point. Such a project switches to `Error`, `ManifestsApplied` names the error, and a create or a change of the project is refused until the text renders. A change of `manifests` reaches every project of the template at once, so try it first on a copy of the template with one project.
- A cluster-scoped object, an object in a namespace that is neither a namespace of the project nor made by a ProjectNamespace of the same `manifests`, an object with the `heritage: multitenancy-manager` or `heritage: deckhouse` label, an object without `metadata.name` and

an object that the text renders twice are refused. While the render holds such an object, none of its objects is applied, the project switches to `Error`, and the `ManifestsApplied` condition names the object.
- An object that exists already and does not belong to the project, for example one that a project administrator created by hand, is not changed, listed or deleted later. The other objects are applied, but nothing is deleted, the project switches to `Error`, `ManifestsApplied` names the object, and the controller looks again every minute. To hand the object over to `manifests`, give it the `projects.deckhouse.io/project: <PROJECT_NAME>` and `projects.deckhouse.io/source: manifests` labels, or delete it so that `manifests` creates it. Only a user who may update objects of its kind in every namespace, such as a cluster administrator, may give it the labels. Do not delete an AuthorizationRule of the `Admin` access level this way, because `manifests` does not create one. Give it the labels instead.
- An object that the `d8:project:admin` role does not allow is refused by the API server with `403 Forbidden`. Examples are a RoleBinding to a role that a project administrator cannot bind and an object of a kind that the role does not cover. The other objects are applied, the project switches to `Error`, `ManifestsApplied` names the refused object, and the controller tries again with a growing interval.
- An AuthorizationRule of the `Admin` access level is refused when `manifests` creates it, raises an existing one to this level or adds subjects to an existing one of this level, although the `d8:project:admin` role allows it. The administrators of a project are set in `.spec.administrators` of the Project. An AuthorizationRule of the `User`, `PrivilegedUser` or `Editor` level is applied as other objects are. The project switches to `Error` as for an object that the role does not allow, and a create or a change of a project whose `manifests` render a refused AuthorizationRule is admitted with a warning that names it. A create of a template whose `manifests` describe an AuthorizationRule, and a change of such `manifests`, are admitted with a warning about this refusal. An AuthorizationRule that exists already, for example one that a switch from a Helm text left in place, is taken over and updated as other objects are, unless `manifests` raises it to the `Admin` level or adds subjects to it at this level.
- An object of a kind that the API server does not serve keeps all the objects from being applied. The project switches to `Error`, and the controller tries again every minute, for example while the module that brings the kind is being enabled.
- `manifests` cannot be set together with `resourcesTemplate`.
- `deckhouse.io/v1alpha1` has no place for `manifests`, so a write through `deckhouse.io/v1alpha1` to a template that has it is refused.
- The `projects.deckhouse.io/skip-heritage-label` and `projects.deckhouse.io/unmanaged` labels of a Helm text have no effect on the objects of `manifests`.

`manifests` does not create cluster-scoped objects. To give the projects of a template access to cluster-wide resources, use `resources` and `grantPolicies`. A cluster administrator creates other cluster-scoped objects outside the template.

Moving a template from a Helm text

A template with a Helm text in `resourcesTemplate` keeps working, but this form is deprecated. `resourcesTemplate` goes away from `deckhouse.io/v1alpha2` together with `deckhouse.io/v1alpha1` in a future release, and no template can get a new one (see Template checks). The `MultitenancyManagerHelmProjectTemplate` alert names every such template, except the built-in ones. Nothing in the cluster rewrites such a template, so move it to structured fields and `manifests` where it is applied from, for example in Git or in CI.

The built-in templates `default`, `secure` and `secure-with-dedicated-nodes` are structured and accept the same parameters as their Helm texts did, except `administrators` and `resourceQuota`, so they show how a text maps to the fields. A field takes the value of a parameter with `{fromParam: <PARAMETER_NAME>}`, so the projects keep their parameters. The following table shows where the objects of such a text go:

Object of the Helm text	Replacement
The `security.deckhouse.io/pod-policy` label of the namespace from the `podSecurityProfile` parameter	`podSecurityStandard: {fromParam: podSecurityProfile}`
The NetworkPolicy objects of the `networkPolicy` parameter	`networkPolicy.mode: {fromParam: networkPolicy}`
The `extended-monitoring.deckhouse.io/enabled` label of the `extendedMonitoringEnabled` parameter	`features.monitoring: {fromParam: extendedMonitoringEnabled}`
The `security-scanning.deckhouse.io/enabled` label of the `securityScanningEnabled` parameter	`features.vulnerabilityScanning: {fromParam: securityScanningEnabled}`

Object of the Helm text	Replacement
The PodLoggingConfig of the `clusterLogDestinationName` parameter	`logShipping.clusterDestinationRef: {fromParam: clusterLogDestinationName}`
The SecurityPolicy of the `allowedUIDs` and `allowedGIDs` parameters	`allowedUIDs: {fromParam: allowedUIDs}` and `allowedGIDs: {fromParam: allowedGIDs}`
The FalcoAuditRules of the `runtimeAuditEnabled` parameter	`runtimeAudit.enabled: {fromParam: runtimeAuditEnabled}`
Other labels and annotations of the namespace	`namespaceMetadata.labels` and `namespaceMetadata.annotations`, in the built-in templates from the `namespace.labels` and `namespace.annotations` parameters
The placement on dedicated nodes	`nodeSelector` and `tolerations`, in the built-in templates from the `dedicatedNodes.nodeSelector` and `dedicatedNodes.defaultTolerations` parameters
The AuthorizationRule objects of the `administrators` parameter	`.spec.administrators` of each Project
Other AuthorizationRule objects of the `Admin` access level	`.spec.administrators` of each Project, because `manifests` does not create an AuthorizationRule of this level
The `all-pods` ResourceQuota of the `resourceQuota` parameter	`.spec.quota` of each Project
The `required-requests-*` OperationPolicy, which refuses pods without CPU and memory requests	The `requiredRequests` parameter, declared with `type: boolean` and `default: true` to keep the policy or with `default: false` to leave it off
Any other object in the namespaces of the project, such as a workload, a ConfigMap, a RoleBinding, a LimitRange or an AuthorizationRule of another access level	`manifests`
A cluster-scoped object	No replacement in a template. Access to cluster-wide resources is set with `resources` and `grantPolicies`

For example, an existing template `team` has the following Helm text, which sets the pod security profile from a parameter and creates a LimitRange in the main namespace of each project. Such a template cannot be created again, so the manifest only shows the starting point:

```
apiVersion: deckhouse.io/v1alpha1
kind: ProjectTemplate
metadata:
  name: team
spec:
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        podSecurityProfile:
          type: string
          enum: [Baseline, Restricted]
          default: Baseline
  resourcesTemplate: |
    apiVersion: v1
    kind: Namespace
    metadata:
      name: {{ .projectName | quote }}
      labels:
        security.deckhouse.io/pod-policy: {{ .parameters.podSecurityProfile | lower |
quote }}
    ---
    apiVersion: v1
    kind: LimitRange
    metadata:
      name: container-defaults
      namespace: {{ .projectName | quote }}
    spec:
      limits:
        - type: Container
          default:
            cpu: 500m
            memory: 512Mi
```

In the structured form, the profile goes to `podSecurityStandard` and the LimitRange goes to `manifests`. The text did not render the `required-requests-*` OperationPolicy, so the template declares the `requiredRequests` parameter with `default: false`:

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: team
spec:
  podSecurityStandard:
    fromParam: podSecurityProfile
  parametersSchema:
    openAPIV3Schema:
      type: object
      properties:
        podSecurityProfile:
          type: string
          enum: [Baseline, Restricted]
          default: Baseline
        requiredRequests:
          type: boolean
          default: false
  manifests: |
    apiVersion: v1
    kind: LimitRange
    metadata:
      name: container-defaults
    spec:
      limits:
        - type: Container
          default:
            cpu: 500m
            memory: 512Mi
```

After the switch, the namespace of each project keeps its label, the LimitRange stays in place, and `manifests` takes it over and removes the annotations of Helm from it.

A template that no project uses can be deleted, and the alert goes with it. If the template is an unchanged copy of a built-in template, move its projects to that template in `.spec.projectTemplateName` instead, and then delete the copy. To move a template that projects use, follow these steps:

1. Find the projects of the template that still set `administrators` or `resourceQuota` in `.spec.parameters`:

   ```
   d8 k get projects -l projects.deckhouse.io/project-template=<PROJECT_TEMPLATE_NAME> -o json | jq -r '.items[] | select((.spec.parameters // {}) | has("administrators") or has("resourceQuota")) | .metadata.name'
   ```

   These values get no standard fields and are not handed over. Move them to `.spec.administrators` and `.spec.quota` first, and only as far as they mean the same there. `.spec.administrators` gives users and groups the `d8:project:admin` role, so an administrator with another access level or another kind of subject needs a ProjectRoleBinding of its own. `.spec.quota` enforces every limit it holds, `0` included, so leave out a limit that the text did not render, such as a `0` that it skipped.

2. Make every tool that writes the template, such as a GitOps tool or a generator, write it as `deckhouse.io/v1alpha2`. Once the template has a field that `deckhouse.io/v1alpha1` cannot describe, such as `manifests`, a write of the template through `deckhouse.io/v1alpha1` is refused, because it would erase that field.

3. Write the template as `deckhouse.io/v1alpha2` with the fields from the table and with `manifests`, without `resourcesTemplate`, in one change. Keep in `parametersSchema` the parameters that the projects set. Declare the `requiredRequests` parameter with `type: boolean`, with `default: true` if the text rendered the `required-requests-*` OperationPolicy and with `default: false` if it did not, so the projects keep refusing or admitting pods without CPU and memory requests as before. A template that projects use cannot lose `resourcesTemplate` without structured fields or `manifests` in its place.

4. Check each project after the switch. Its `HandoverPending` condition is removed once the replaced

objects are deleted, its `ManifestsApplied` condition says whether the objects of `manifests` are applied, and `.status.keptObjects` lists the objects left in place:

```
d8 k get project <PROJECT_NAME> -o jsonpath='{range .status.conditions[*]}{.type}=
{.status}: {.message}{"\n"}{end}{range .status.keptObjects[*]}{.kind}
{.namespace}/{.name}{"\n"}{end}'
```

5. Decide on each object that stays in `.status.keptObjects`, as Switching a project from a Helm text describes. An object that `manifests` renders under the same kind, namespace and name leaves the list once `manifests` takes it over, and from then on it has what `manifests` renders.

Switching a project from a Helm text

A project release rendered from a Helm text in `resourcesTemplate` holds whatever the text rendered. The project switches to structured fields when its template gets structured fields or `manifests` in place of `resourcesTemplate`, when the project moves to a structured template, and when a built-in template that was a Helm text is rewritten with structured fields. The release then holds only what the structured fields render. An object of the Helm text that the structured fields render under the same kind and name stays in the release and is updated in place. The other objects of the Helm text are handled as follows, and of them only an AuthorizationRule that names nobody from `.spec.administrators` any more is deleted at once.

The AuthorizationRule and ResourceQuota objects that the standard fields replace go through a handover, so the administrators never lose access and the namespace never loses its quota. These are the AuthorizationRule objects the release created that grant the `Admin` access level and nothing else to users and groups who are all administrators of the project, and the ResourceQuota objects of the main namespace without scopes whose every limit `.spec.quota` sets to the same value or a lower one. The handover goes in the following order:

1. The Project gets the `HandoverPending` condition set to `True`. Then the objects get the `helm.sh/resource-policy: keep` annotation and the `projects.deckhouse.io/kept-by-handover: "true"` annotation, so the release upgrade does not delete them.
2. The `d8-administrators` ProjectRoleBinding and the `d8-project-quota` ResourceQuota are created before the release is upgraded.
3. The kept objects are deleted once `d8-administrators` names the administrators from `.spec.administrators` and is `Ready` for its current `metadata.generation`, the `d8:project:admin` ClusterRole has its aggregated rules, and `d8-project-quota` has the limits from `.spec.quota` and is calculated, and not earlier than 2 minutes after `d8-administrators` became `Ready` and after the handover began. Until then the administrators hold both roles, and the tighter of the quotas applies. The `HandoverPending` condition says what the handover waits for, and it is removed once the kept objects are deleted.

A kept AuthorizationRule that names nobody from `.spec.administrators` any more grants only what was revoked, and it is deleted at once. A subject removed from `.spec.administrators` is removed at once from a kept AuthorizationRule that names other administrators too. A kept quota that `.spec.quota` no longer covers when the handover finishes, because a limit in `.spec.quota` was raised above it or removed, is left in place like the objects below. The `HandoverPending` condition is kept in the status of the Project, so a replace of the Project (for example, `kubectl replace` or Argo CD with the `Replace=true` option) does not lose it. If the kept objects are still there after 30 minutes, the `MultitenancyManagerProjectHandoverStuck` alert names the project and says what to check.

Every other object of the Helm text that the structured fields do not render is left in place instead of being deleted. Examples are AuthorizationRule objects of other subjects or of another access level, such as the ones a text renders for roles from its own parameters, a quota that `.spec.quota` does not cover, RoleBinding objects, network policies, and workloads with their volumes.

- Such an object keeps working as before. A rule or a binding keeps granting access, a quota keeps applying, a workload keeps running.
- The project no longer manages it. The object gets the `helm.sh/resource-policy: keep` annotation and the `projects.deckhouse.io/kept-from-helm-template: "true"` label, and it loses the `heritage: multitenancy-manager` label. While it carries the label, only a requester who may make the same change to objects of its kind across the cluster may change or delete it, for example a cluster administrator. The users of the project may not, so a network policy or a quota of the text keeps holding them. The protection does not cover the `scale` and `status` subresources: whoever may scale objects of the kind or change their status in the namespace may still do so for a kept object, which matters only for a kept workload.
- The Project lists it in `.status.keptObjects`, and the `MultitenancyManagerProjectKeepsHelmTemplateObjects` alert names the project. An object leaves the list once it is deleted or the label is removed from it. When the template renders an object of the same kind and name again, the project release takes the object over. The controller then removes the label, puts the `heritage: multitenancy-manager` label back and removes the

`helm.sh/resource-policy: keep` annotation, unless the template sets it itself.
- When `manifests` of the template renders an object of the same kind, namespace and name, the controller applies it as the user of the project and takes the object over. The object then has what `manifests` renders. A field that the Helm text set and `manifests` does not set is removed, the annotations of Helm included, such as `helm.sh/resource-policy: keep` and `meta.helm.sh/*`, and so is the `projects.deckhouse.io/kept-from-helm-template` label. A field that somebody else set on the object stays. The object leaves `.status.keptObjects`.
- When the project is deleted, the object goes with it. An object in a namespace of the project goes with the namespace, and the controller deletes a cluster-scoped object that still carries the label.

Describe what is still needed in the template or in the project, for example with a ProjectRoleBinding or with a limit in `.spec.quota`, and delete the left object, or remove the label to hand the object over to the project, after which the users of the project may change and delete it as well. An object in a namespace of the project can instead be described in `manifests` of the template, which takes it over. A cluster-scoped object without the label also stays when the project is deleted. An access that a left object grants ends only when the object is deleted. To list the left objects of a project, use the command:

```
d8 k get project <PROJECT_NAME> -o jsonpath='{range .status.keptObjects[*]}{.kind}
{.namespace}/{.name}{"\n"}{end}'
```

An edit of a Helm text, and a move of a project to another template with a Helm text, delete the objects the new text does not render, except the AuthorizationRule and ResourceQuota objects that the standard fields replace. These go through the same handover.

Administrators and a quota that are still set in `.spec.parameters`, because they could not be moved to `.spec.administrators` and `.spec.quota`, get no standard fields. Their AuthorizationRule objects and their quota are not handed over, and a switch leaves them in place. Move such values to the standard fields before the switch.

After the handover the administrators have the `d8:project:admin` role in place of the `Admin` access level. The two roles do not grant the same permissions. In particular, the access that custom ClusterRoles with the `user-authz.deckhouse.io/access-level` annotation added to the `Admin` access level is not part of `d8:project:admin`. To grant such access, create a custom role and bind it with a ProjectRoleBinding.

Moving an object of the release to manifests

A structured template renders its objects into the project release. When one change makes `manifests` render an object the release contained, under the same kind, namespace and name, the upgrade would delete the object and `manifests` would create it again. The project would be without the object in between. An example is the NetworkPolicy named `isolated` once `networkPolicy.mode` is removed and `manifests` render that NetworkPolicy.

The controller leaves the object in place instead. Before the upgrade it sets the `helm.sh/resource-policy: keep` annotation and the `projects.deckhouse.io/kept-from-helm-template: "true"` label, and lists the object in `.status.keptObjects`, the same way a switch from a Helm text leaves an object. In the same reconcile `manifests` apply the object as the user of the project, remove the label and take the object off the list. The object then has what `manifests` render. A field that the release set and `manifests` do not set is removed. A field that somebody else set stays.

If `manifests` cannot apply the object, for example because the API server refuses the kind or returns `403 Forbidden`, the object stays in `.status.keptObjects`. The `MultitenancyManagerProjectKeepsHelmTemplateObjects` alert names the project. The object keeps working, and only a requester who may make the same change to objects of its kind across the cluster may change or delete it, until `manifests` take it over or the label is removed.

An object the release drops and that `manifests` do not render is deleted by the upgrade.

Creating your own project template

To create your own template:

1. Take one of the default templates as a basis, for example, `default`.
2. Copy it to a separate file, for example, `my-project-template.yaml` using the command:

   ```
   d8 k get projecttemplates default -o yaml > my-project-template.yaml
   ```
3. Edit the `my-project-template.yaml` file and adjust the structured fields, the objects in `manifests` and the input parameters schema to your needs.

4. Change the template name in the `.metadata.name` field.
5. Remove the `heritage` label, the `metadata.resourceVersion`, `metadata.uid`, `metadata.creationTimestamp`, `metadata.generation` and `metadata.managedFields` fields and the `status` section from the file.
6. Apply your new template with the command:

   ```
   d8 k apply -f my-project-template.yaml
   ```

7. Check the availability of the new template with the command:

   ```
   d8 k get projecttemplates <NEW_TEMPLATE_NAME>
   ```

Labels for the objects of a Helm text

The following labels change how the module handles the objects of a Helm text in `resourcesTemplate`. Only existing templates have such a text, because no template can get a new `resourcesTemplate`. The objects of `manifests` never get the `heritage: multitenancy-manager` label, and the module does not read these labels on them.

Skipping the heritage label

By default, every object of the Helm text gets the `heritage: multitenancy-manager` label. With this label, only the identities listed in Who bypasses admission can change the object. To allow changes to the object (for example, for compatibility with other systems, or to control the created objects yourself), add the `projects.deckhouse.io/skip-heritage-label` label to it.

Example:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/skip-heritage-label: "true"
    app: my-app
data:
  key: value
```

In this case, the object gets the `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` labels, but not the `heritage: multitenancy-manager` label.

Excluding objects from management

To exclude an object from management by the module (for example, when it is managed manually or by another controller), add the `projects.deckhouse.io/unmanaged` label to it.

Example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Objects with the `projects.deckhouse.io/unmanaged` label:

- are created **only once**, when the project release is first installed, so an object added to the template later is not created in existing projects;
- **are not updated** when the template changes;
- are not tracked in the project status;
- get the `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` labels, but **not** the `heritage: multitenancy-manager` label;
- get the `helm.sh/resource-policy: keep` annotation, so the project release does not delete them.

> ⚠ After it is created, an `unmanaged` object does not depend on the template and must be managed manually.

Implementing validation of object changes with a custom label

The `multitenancy-manager` module uses a ValidatingAdmissionPolicy to protect objects labeled `heritage: multitenancy-manager` from manual changes. You can implement similar validation for objects with any label.

How validation works in multitenancy-manager

Validation occurs for objects labeled `heritage: multitenancy-manager`. The following components are used for this:

1. ValidatingAdmissionPolicy: Defines validation rules:
   - Operations: `CREATE`, `UPDATE` and `DELETE`. `system:masters` is not exempt.
   - Check: only the module's controller and the identities this policy lets through (for details, see Who bypasses admission) may change such an object. For a project namespace, an update that changes only labels and annotations outside the keys the module owns is allowed as well (see Creating a project automatically for a namespace).
   - Applies to all resources and API groups.
2. ValidatingAdmissionPolicyBinding: Defines which objects the validation applies to:
   - Uses `namespaceSelector` and `objectSelector` to select resources by the label `heritage: multitenancy-manager`.

Who bypasses admission

The webhooks and policies of the module have different exclusion lists:

- The **Project and ProjectTemplate webhooks** skip requests from platform components — the API server (`system:apiserver`), the service accounts of Deckhouse (`d8-system:deckhouse`), of this module (`d8-multitenancy-manager:multitenancy-manager`) and of `user-authz` (`d8-user-authz:controller`), every service account of the `d8-system`, `kube-system` and `d8-user-authz` namespaces, kubelets (`system:nodes`) — and from the `system:sudouser` identity. Everyone else is validated, `system:masters` included.
- The **ProjectRoleBinding, ProjectNamespace and ClusterProjectRoleBinding webhooks** skip nobody and validate every request. Inside them only the controller and Deckhouse service accounts are privileged; `system:sudouser` and `system:masters` are ordinary users there.
- The **ValidatingAdmissionPolicy** above lets through this module's controller; the users `system:apiserver`, `system:kube-controller-manager`, `system:kube-scheduler`, `system:volume-scheduler`, `dhctl`, `observability` and `system:sudouser`; the groups `system:nodes`, `system:serviceaccounts:kube-system` and `system:serviceaccounts:d8-system` (not `d8-user-authz`); and a rollout restart of a project workload that changes only the `kubectl.kubernetes.io/restartedAt` annotation of the pod template. Everyone else is refused, `system:masters` included.
- The **`d8-multitenancy-manager-manifests-objects` ValidatingAdmissionPolicy** guards objects labelled `projects.deckhouse.io/source: manifests`. It lets through this module's controller; the users `system:apiserver`, `system:kube-controller-manager`, `system:kube-scheduler`, `system:volume-scheduler`, `dhctl`, `observability` and `system:sudouser`; and the groups `system:nodes`, `system:serviceaccounts:kube-system` and `system:serviceaccounts:d8-system`. Every service account whose name starts with `system:serviceaccount:d8-` may create, change and delete such objects. The user `system:multitenancy-manager:project:<PROJECT_NAME>` may create, change and delete the objects of that project. A rollout restart that changes only the `kubectl.kubernetes.io/restartedAt` annotation of the pod template is allowed. An update that puts the `projects.deckhouse.io/source: manifests` label, together with the `projects.deckhouse.io/project` label, on an object that does not have the `projects.deckhouse.io/source: manifests` label yet is allowed to a requester who may

update that resource across the cluster. Everyone else is refused, `system:masters` included.
- The **ValidatingAdmissionPolicy that reserves the users the controller acts as** (the names that start with `system:multitenancy-manager:`, see Describing other objects in manifests) lets through this module's controller, the garbage collector, which runs as `system:serviceaccount:kube-system:generic-garbage-collector` or as `system:kube-controller-manager`, and `system:sudouser`. Everyone else who names such a user or group in a binding is refused, the other service accounts of `kube-system` and `system:masters` included.
- The **`d8-multitenancy-manager-namespace-creation` ValidatingAdmissionPolicy** checks namespace creation and is in force only while `allowNamespacesWithoutProjects` is `false`. It lets through the API server (`system:apiserver`), the service accounts of Deckhouse (`d8-system:deckhouse`), of this module (`d8-multitenancy-manager:multitenancy-manager`) and of the `upmeter` agent (`d8-upmeter:upmeter-agent`), every service account of the `d8-system` and `kube-system` namespaces, the groups `system:masters` and `system:nodes`, and `system:sudouser`. Unlike the lists above, it includes `system:masters`, so cluster administrators still create namespaces directly. Everyone else can create only the `default` namespace.
- The **`d8-multitenancy-manager-namespace-adoption` ValidatingAdmissionPolicy** refuses to create a namespace whose name is longer than 61 characters, starts with `upmeter-` or is `deckhouse`, and to put the `heritage: upmeter` label on a namespace, whatever `allowNamespacesWithoutProjects` says. It lets through the same users and groups as the `d8-multitenancy-manager-namespace-creation` policy.
- The **`d8-multitenancy-manager-manifests` ValidatingAdmissionPolicy** checks only the users that the controller applies `manifests` as, whose names start with `system:multitenancy-manager:project:`, and refuses them to create an AuthorizationRule of the `Admin` access level, to raise one to this level or to add subjects to one of this level. Nobody else is checked by it.
- The **`d8-multitenancy-manager-v1alpha1-template-create` ValidatingAdmissionPolicy** refuses every create of a ProjectTemplate through `deckhouse.io/v1alpha1` and lets nobody through, `system:sudouser`, `system:masters` and the platform components included. The refusal of a new `resourcesTemplate` through `deckhouse.io/v1alpha2` belongs to the ProjectTemplate webhook, so the requesters that the webhook skips, `system:sudouser` among them, can still create a template with a Helm text that way.
- The **`d8-multitenancy-manager-namespace-ownership` ValidatingAdmissionPolicy** keeps the `projects.deckhouse.io/project` label and the labels and annotations that the module sets on a project namespace. It lets through only this module's controller and `d8-system:deckhouse`. Everyone else is refused, `system:sudouser` and `system:masters` included.
- The **`d8-multitenancy-manager-project-grants` ValidatingAdmissionPolicy** keeps the ClusterResourceGrantPolicy objects that the controller makes from `spec.grants` of a Project. It refuses to create an object whose name starts with `project-`, and to create, change or delete an object that has the `multitenancy.deckhouse.io/managed-by-project` label. It lets through this module's controller, `d8-system:deckhouse`, `system:sudouser` and the service accounts of the `kube-system` namespace, the garbage collector among them. Everyone else is refused, `system:masters` included.
- The **`d8-multitenancy-manager-kept-objects` ValidatingAdmissionPolicy** guards the objects listed in `.status.keptObjects` of a project. It lets through this module's controller, the users whose names start with `system:multitenancy-manager:`, `system:sudouser` and the system users and groups of the policy that guards the `heritage: multitenancy-manager` objects. Anyone else may change such an object only with the right to make the same change across the cluster.

The `/is-granted` and `/defaults` webhooks, which check writes of objects that reference cluster-wide resources in project namespaces (see Managing access to cluster-wide resources), skip `system:masters` together with the platform components. The `/protect` webhook keeps AvailableClusterResource read-only. It lets through the controller, the platform components and `system:masters`.

An administrator who needs to bypass the Project or ProjectTemplate webhook, the policy that guards the `heritage: multitenancy-manager` objects, the `d8-multitenancy-manager-manifests-objects` policy, or the policy that reserves the users the controller acts as, does it deliberately as `system:sudouser`, for example with `d8 k --as system:sudouser …`, and the bypass is then visible in the audit log. There is no similar option for ProjectRoleBinding, ProjectNamespace and ClusterProjectRoleBinding, for a create of a ProjectTemplate through `deckhouse.io/v1alpha1`, or for the namespace labels and annotations that the `d8-multitenancy-manager-namespace-ownership` policy keeps.

Creating your own validation

To implement validation for resources with a different label (for example, `heritage: my-custom-label`):

1. Create a file with the ValidatingAdmissionPolicy and ValidatingAdmissionPolicyBinding resource manifests:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:   ["*"]
        apiVersions: ["*"]
        operations:  ["CREATE", "UPDATE", "DELETE"]
        resources:   ["*"]
        scope: "*"
  matchConditions:
    # Let the cluster components through, as the policy of the module does.
    - name: exclude-system-groups
      expression: '!(["system:nodes", "system:serviceaccounts:kube-system",
"system:serviceaccounts:d8-system"].exists(e, (e in request.userInfo.groups)))'
    - name: exclude-system-users
      expression: '!(["system:apiserver", "system:kube-controller-manager",
"system:kube-scheduler"].exists(e, (e == request.userInfo.username)))'
  variables:
    # A DELETE request carries no object, only oldObject.
    - name: subject
      expression: 'object == null ? oldObject : object'
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-
namespace:my-service-account"' # Replace with your ServiceAccount.
      reason: Forbidden
      messageExpression: 'variables.subject.kind == ''Namespace'' ? ''This
resource is managed by '' + variables.subject.metadata.name + '' system. Manual
modification is forbidden.''
       : ''This resource is managed by '' + variables.subject.metadata.namespace
+ '' system. Manual modification is forbidden.'''
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    objectSelector:
      matchLabels:
        heritage: my-custom-label
```

2. Configure the validation parameters:
   - `policyName`: Unique policy name (must match in Policy and Binding).
   - `request.userInfo.username`: The name of the ServiceAccount allowed to change objects (replace with your ServiceAccount).
   - `heritage: my-custom-label`: The value of the `heritage` label for your resources (replace with your value). The use of the values `multitenancy-manager`, `deckhouse` is prohibited.
   - `failurePolicy`: What happens when the policy cannot be evaluated, for example when an expression fails: `Fail` rejects the request, `Ignore` admits it. A validation that returns `false` rejects the request with the `Deny` action in any case.
   - `matchConditions`: Requests that the policy does not check. Keep the cluster components in this list, otherwise the policy blocks the deletion of namespaces and the garbage collection of the labeled objects.
   - `validationActions`: Validation actions:
     - `Deny`: Deny unauthorized operations.
     - `Audit`: Record operations in the audit log.

3. Apply the policy:

3. Apply the policy:

```
d8 k apply -f my-validation-policy.yaml
```

4. Ensure your resources have the corresponding `heritage` label:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label
```

Managing access to cluster-wide resources

The module allows you to manage project access to cluster-wide resources such as StorageClass, ClusterIssuer, ClusterRole, and others.

For a description of the mechanism, the resources it uses, and the cluster-wide resources registered by the platform, refer to the module description.

The following sections provide common scenarios for configuring and using the mechanism.

How a policy selects namespaces

A ClusterResourceGrantPolicy (`multitenancy.deckhouse.io/v1alpha2`) has two selectors, `projectSelector` and `namespaceSelector`. The policy applies to a namespace of a project when every selector it sets matches:

- `projectSelector` is matched against the labels of the Project object. It selects every namespace of a project at once, including an additional namespace created later, so use it to select a project as a whole. The module puts the `projects.deckhouse.io/project: <PROJECT_NAME>` and `projects.deckhouse.io/project-template: <PROJECT_TEMPLATE_NAME>` labels on every Project, so a project can also be selected by its name or template. The virtual projects carry only the first of these labels. The two labels cannot be set to another project or template. A manifest that replaces a whole Project may leave them out. The request is admitted with a warning, and the module puts the labels back. The template label of a Project changes as soon as `spec.projectTemplateName` changes. The same label on the namespaces shows the template they were rendered from, and the service policies of a template select namespaces by it.
- `namespaceSelector` is matched against the labels of the Namespace. It narrows the choice to some namespaces of the selected projects, or of all projects when `projectSelector` is not set. For example, it can select only the main namespace (the `projects.deckhouse.io/project-namespace` label does not exist) or only the namespaces labeled by a template.
- A policy with **neither** selector matches no project on its own. Such a policy is a library policy, applied to a project through `grantPolicies` of its ProjectTemplate or of the Project itself (`spec.grants.grantPolicies`). An explicit empty selector (`{}`) matches everything.

The namespaces of the virtual projects `deckhouse` and `default` are never covered, even by `{}`.

An additional namespace is created by a project user and carries only the labels the module sets and the labels of the project template. A policy that restricts a resource through `namespaceSelector` may therefore not cover a new additional namespace. This happens when the selector relies on a label set on a namespace by hand, on a template label whose value was changed by hand on the main namespace, or on `projects.deckhouse.io/project-namespace`, whose value is the name the user chooses. If the GrantableClusterResourceDefinition of the resource has `defaultAvailability: All` (the default), such a namespace gets access to every resource of that type.

The policy reports this in its `NewNamespacesCovered` condition. The condition checks `namespaceSelector` against the labels that a new additional namespace of each covered project would get. Denied names count as a restriction for any `defaultAvailability` value. An allow-list or `availabilityDefault: None` counts only when `defaultAvailability` is `All`.

Do not select namespaces by the `projects.deckhouse.io/project-namespace` label without a `projectSelector`. A user of any project can create a ProjectNamespace with the name that such a `namespaceSelector` expects. The policy then applies in that project and makes the resources it allows available there.

The `multitenancy.deckhouse.io/v1alpha1` version is still served. Its `projectSelector` is matched against the labels of the namespace and corresponds to `namespaceSelector` of `v1alpha2`. `v1alpha1` has no field for the `v1alpha2` `projectSelector`, so a `v1alpha1` client sees it as JSON in the

`multitenancy.deckhouse.io/v1alpha2-project-selector` annotation. A policy with this annotation selects projects by it, even though its `v1alpha1` `projectSelector` is empty.

To move a policy manifest from `v1alpha1` to `v1alpha2`, rename its `projectSelector` to `namespaceSelector`. Do not change only `apiVersion`. The same `projectSelector` is then matched against the labels of the Project, so a selector by a label that only the namespace has, such as `kubernetes.io/metadata.name` or a label from `namespaceMetadata`, matches nothing, and the policy silently stops applying to those namespaces.

When you change such a policy through `v1alpha1`, keep the annotation. `d8 k apply` and `d8 k edit` keep it, while `d8 k replace` with a manifest that lacks the annotation removes the `projectSelector`. To return to an old `v1alpha1` manifest after you set the `v1alpha2` `projectSelector`, remove the annotation explicitly, because `d8 k apply` of the old manifest leaves it in place.

For cluster administrators

The following examples show how to configure project access to cluster-wide resources using ClusterResourceGrantPolicy. They select projects with `projectSelector` by a label on the Project, which covers every namespace of the project.

Because the labels of a Project and of its namespaces take part in the match, the right to change these labels decides which policies apply. The labels of a Project are changed with the `d8:system-capability:multitenancy-manager:edit` capability, which the `iam` subsystem manager has and which also allows creating and changing the policies. The labels of a namespace are changed with the `d8:subsystem-capability:cluster:manage` `resources` capability, which the `cluster` subsystem manager has. The project roles `d8:project:*` and `d8:namespace:*` only read namespaces.

Restricting StorageClass for a project

To allow projects to use only the `fast-ssd` and `standard` StorageClasses and use `fast-ssd` by default, create the following ClusterResourceGrantPolicy:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: production-storage
spec:
  projectSelector:
    matchLabels:
      environment: production
  resources:
    - resourceName: storageclasses
      default: fast-ssd
      allowed:
        - fast-ssd
        - standard
```

When a PersistentVolumeClaim is created without a value in `spec.storageClassName`, `fast-ssd` is automatically assigned to this field. This holds even if the built-in Kubernetes admission controller has already filled in the default StorageClass of the cluster, for example `standard`: StorageClass uses the `Coerce` default assignment mode, which replaces a value the author of the object did not set. If a StorageClass that is not in the allowed list is specified, it is replaced with `fast-ssd` as well. This replacement is reported as an admission warning in the response to the request, so the `d8 k` output shows both the original value and the substituted one. A PersistentVolumeClaim with `storageClassName: ""`, which asks for a volume without a class, keeps the empty value. If the project has no default StorageClass, a PersistentVolumeClaim with a StorageClass that is not in the allowed list is rejected.

A PersistentVolumeClaim created without a class while neither the project nor the cluster has a default StorageClass stays without a class. Once the cluster gets a default StorageClass, Kubernetes (kube-controller-manager) assigns it to such a claim, and the grant webhooks do not check this change, so the claim can get a class the project may not use. Set `default` in the grant policy to avoid it.

A PersistentVolumeClaim that restores a volume from a snapshot or clones another volume (`spec.dataSource` or `spec.dataSourceRef` with `kind: VolumeSnapshot` or `kind: PersistentVolumeClaim`) keeps its StorageClass: the class must be served by the CSI driver of the source, so it is not replaced with the project default. If such a StorageClass is not available to the project, the PersistentVolumeClaim is rejected with an explanation; specify a StorageClass that is

available to the project.

A PersistentVolumeClaim with the `volume.beta.kubernetes.io/storage-class` annotation is checked by the class in the annotation, not by `spec.storageClassName`: Kubernetes takes the class from the annotation first. If that class is not available to the project, the PersistentVolumeClaim is rejected and the class is not replaced with the project default; name a StorageClass that is available to the project in the annotation, or remove the annotation and set `spec.storageClassName`.

A PersistentVolumeClaim created from a template is checked in the template: kube-controller-manager creates such claims on behalf of a system account whose requests are not checked. So the StorageClass is checked in the volume claim templates of StatefulSets (`spec.volumeClaimTemplates`) and in the ephemeral volumes (`ephemeral.volumeClaimTemplate`) of pods, Deployments, ReplicaSets, DaemonSets, StatefulSets, Jobs, CronJobs and ReplicationControllers. The grant webhooks receive every object of these kinds in a project namespace for the PriorityClass and RuntimeClass checks, so while multitenancy-manager does not answer, creating or changing such an object fails whether or not it has such a template.

Like Kubernetes, the check takes the class from the `volume.beta.kubernetes.io/storage-class` annotation of a template if it is set, and from `spec.storageClassName` otherwise. A class in a template is not replaced with the project default: a StorageClass that is not available to the project is rejected. The volume of a template without a StorageClass gets the default StorageClass of the cluster, not the project default, so such a template is rejected if the default StorageClass of the cluster is not available to the project, and the rejection names that class. If the cluster has no default StorageClass, such a template is admitted, and its claim gets the default StorageClass once the cluster has one, without a check. An empty value (`storageClassName: ""`) binds a pre-provisioned PersistentVolume and is allowed. Objects that already had such templates can still be updated while the templates do not change.

To check which StorageClasses are available to the project, run the following command:

```
d8 k get available storageclasses -n <PROJECT_NAME> -o yaml
```

> **i** While the `storage-foundation` module of version 1.0.3 or later is enabled, the `storageclasses` definition and its GrantableClusterResourceReference objects are registered by that module and can differ from the ones described here, see the documentation of that module.

Restricting ClusterIssuer for a project

To allow projects to use only the `letsencrypt-prod` and `vault-issuer` ClusterIssuers and use `letsencrypt-prod` by default, create the following ClusterResourceGrantPolicy:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: production-issuers
spec:
  projectSelector:
    matchLabels:
      environment: production
  resources:
    - resourceName: clusterissuers
      default: letsencrypt-prod
      allowed:
        - letsencrypt-prod
        - vault-issuer
```

The policy applies to a ClusterIssuer specified in either of the following ways:

- In the `.spec.issuerRef.name` field of a Certificate if `.spec.issuerRef.kind` is set to ClusterIssuer.
- In the `cert-manager.io/cluster-issuer` annotation of an Ingress.

When a Certificate with a ClusterIssuer is created without a value in `.spec.issuerRef.name`, `letsencrypt-prod` is automatically assigned to this field. If a ClusterIssuer that is not in the allowed list is specified, the Certificate is rejected.

No default value is assigned to the `cert-manager.io/cluster-issuer` annotation. If the annotation is specified, its value is checked against the same list of allowed ClusterIssuers.

> **i** Access management for ClusterIssuer is available only when the `cert-manager` module is enabled: the `clusterissuers` definition is shipped by that module.

Restricting ClusterRoles for a project

By default, a RoleBinding in a project namespace can reference every ClusterRole with the `rbac.deckhouse.io/delegatable: "true"` label, and no other ClusterRole. A policy cannot make a ClusterRole without this label available. To use such a ClusterRole in projects, add the label to it (see Roles available in a RoleBinding inside a project).

To allow the projects of the `payments` team to use only the `my-custom-role` ClusterRole and the ClusterRoles with the `shared: "true"` label, create the following ClusterResourceGrantPolicy:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: extra-roles
spec:
  projectSelector:
    matchLabels:
      team: payments
  resources:
    - resourceName: clusterroles
      allowed:
        - my-custom-role
      allowedSelector:
        matchLabels:
          shared: "true"
```

In RoleBindings of these projects, only these ClusterRoles can be used, and only the ones with the `rbac.deckhouse.io/delegatable: "true"` label. The other delegatable ClusterRoles, such as `d8:project:*` and `d8:namespace:*`, become unavailable in them, including the RoleBindings that the `manifests` of the template create. So list in the policy every role that these projects bind. ProjectRoleBinding and ClusterProjectRoleBinding are not affected, because the controller creates their RoleBindings.

The `rbac.deckhouse.io/delegatable: "true"` label is required on every ClusterRole the policy names: the `clusterroles` registration excludes roles without it from all projects, and an exclusion takes precedence over `allowed` and `allowedSelector`. A policy entry that names a role without the label grants nothing. The controller reports this in the `AllowedEffective` condition of the ClusterResourceGrantPolicy status, with the role name in the message:

```
d8 k get clusterresourcegrantpolicy extra-roles -o jsonpath='{.status.conditions[?
(@.type=="AllowedEffective")]}'
```

When a RoleBinding is created or modified, the ClusterRole specified in it is checked for availability to the project. No ClusterRole value is assigned automatically.

Restricting LoadBalancerClass for a Service

To allow projects to use only the `internal-lb` and `edge-lb` LoadBalancerClass values and use `internal-lb` by default, create the following ClusterResourceGrantPolicy:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: lb-classes
spec:
  projectSelector:
    matchLabels:
      environment: staging
  resources:
    - resourceName: loadbalancerclasses
      default: internal-lb
      allowed:
        - internal-lb
        - edge-lb
```

Unlike StorageClass, ClusterIssuer, and ClusterRole, LoadBalancerClass is not a separate Kubernetes resource. The policy defines the allowed values of the `.spec.loadBalancerClass` field for Services of the `LoadBalancer` type.

When a Service of the `LoadBalancer` type is created without a value in `.spec.loadBalancerClass`, `internal-lb` is automatically assigned to this field. If a value that is not in the allowed list is specified, the Service is rejected.

The policy does not apply to Services of other types.

Granting access to all resources of a specific type

To allow specific projects to use all resources of a selected type without listing them explicitly, set `availabilityDefault: All`.

The following policy allows all projects with the `environment: sandbox` label to use any StorageClass:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: open-storage-for-sandbox
spec:
  projectSelector:
    matchLabels:
      environment: sandbox
  resources:
    - resourceName: storageclasses
      availabilityDefault: All
```

The registered resources are available to all projects by default, so `availabilityDefault: All` changes something only where another policy restricts the resource. There it overrides the `allowed` and `allowedSelector` lists of every policy that applies to the project, the policies of its template included. `denied`, `deniedSelector` and the `excluded` filters of the registration still apply. The resources that match the `reserved` filters of the registration stay unavailable, see “Granting a reserved resource”.

Usually, explicitly specifying allowed resources using `allowed` or `allowedSelector` is sufficient for access management. Use `availabilityDefault: All` when the selected projects need access to all resources of the specified type.

Granting a reserved resource

A GrantableClusterResourceDefinition can mark some resources as reserved with `reserved`. A reserved resource is unavailable to projects by default, even when the other resources of its type are available. It becomes available to a project only if a policy that applies to the project lists it in `allowed` or matches it with `allowedSelector`. `availabilityDefault: All` and `defaultAvailability` do not make it available. `denied`, `deniedSelector` and the `excluded` filters take precedence over `allowed` and `allowedSelector` here as well.

An `allowedSelector` opens every reserved resource it matches. A broad or negative selector (`Exists`, `NotIn`, `DoesNotExist`) written for ordinary resources therefore also grants the reserved resources that happen to match it. To grant a reserved resource, name it in `allowed` or use a selector that targets it explicitly.

In the following examples, the `myclusterresources` registration reserves the `premium` and `internal` resources.

To grant a reserved resource without narrowing the other resources of the type, list it in `allowed` and set `availabilityDefault: All`:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: grant-premium
spec:
  projectSelector:
    matchLabels:
      tier: critical
  resources:
    - resourceName: myclusterresources
      allowed:
        - premium
      availabilityDefault: All
```

The selected projects get every resource of the type that is not reserved, and `premium`. The `internal` resource stays unavailable, because `availabilityDefault: All` does not open reserved resources. `availabilityDefault: All` also overrides the `allowed` and `allowedSelector` lists of the other policies that apply to the project (see “Granting access to all resources of a specific type”).

`allowed` without `availabilityDefault` makes every resource not on its list unavailable, as in any other policy. If a project must stay restricted to a list, for example by the policy of its template, grant the reserved resource with `allowed` alone: the `allowed` lists of all policies that apply to the project add up. Alternatively, list in one policy every resource the project needs, the reserved one included.

To make a reserved resource the project default, specify it in both `allowed` and `default` of the same entry. A default that the project may not use is ignored. This also applies to a reserved resource marked as the default by `defaultFrom`: it becomes the project default only where a policy allows it.

The table shows what a project gets for typical combinations of policies. The registration reserves `premium` and `internal`, excludes `system` with `excluded`, and has `defaultAvailability: All`. The `premium` resource carries the `tier: critical` label. “Policies of the project” lists the entries of all policies that apply to the project.

Policies of the project	standard	premium (reserved)	internal (reserved)	system (excluded)
None	available	—	—	—
`availabilityDefault: All`	available	—	—	—
`allowed: [premium]`	—	available	—	—
`allowed: [premium]` and `availabilityDefault: All`	available	available	—	—
`allowedSelector` matching `tier: critical`	—	available	—	—
`allowed: [standard]` in the template policy and `allowed: [premium]` in another policy	available	available	—	—

Policies of the project	standard	premium (reserved)	internal (reserved)	system (excluded)
`allowed: [premium]` and, in another policy, `denied: [premium]`	—	—	—	—
`allowed: [system]`	—	—	—	—
`allowed: [system]` and `availabilityDefault: All`	available	—	—	—

Denying individual resources

To prevent projects from using individual resources while keeping the others available, use `denied` or `deniedSelector`.

The following policy prevents projects with the `environment: dev` label from using the `expensive-nvme` and `archived-hdd` StorageClasses:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: deny-expensive-storage
spec:
  projectSelector:
    matchLabels:
      environment: dev
  resources:
    - resourceName: storageclasses
      denied:
        - expensive-nvme
        - archived-hdd
```

Deny rules take precedence over allow rules. If a resource matches both `denied` or `deniedSelector` and `allowed` or `allowedSelector`, it is considered unavailable.

Managing access using label selectors

To manage access to resources without listing their names, use `allowedSelector` and `deniedSelector`. Selectors allow or deny resources based on their labels.

The following policy allows projects with the `tier: shared` label to use StorageClasses with the `shared: "true"` label, except for StorageClasses with the `deprecated: "true"` label:

```
apiVersion: multitenancy.deckhouse.io/v1alpha2
kind: ClusterResourceGrantPolicy
metadata:
  name: shared-storage-only
spec:
  projectSelector:
    matchLabels:
      tier: shared
  resources:
    - resourceName: storageclasses
      allowedSelector:
        matchLabels:
          shared: "true"
      deniedSelector:
        matchLabels:
          deprecated: "true"
```

Granting cluster-wide resources through a project template

The availability rules for cluster-wide resources can be set directly in a structured template — they then automatically apply to all projects created from that template:

- `spec.resources`: Rules inside the template, in the same format as `resources` in a ClusterResourceGrantPolicy (resource name, `allowed`/`allowedSelector`, `default`).
- `spec.grantPolicies`: List of names of **library** ClusterResourceGrantPolicy objects. A library policy describes a reusable set of rules and sets neither `projectSelector` nor `namespaceSelector`, because the referencing template determines which projects it applies to. This way, for example, a “corporate StorageClasses” policy can be maintained by one administrator and used by several templates.

The following template restricts StorageClass for its projects and connects a library policy:

```
apiVersion: deckhouse.io/v1alpha2
kind: ProjectTemplate
metadata:
  name: my-template
spec:
  resources:
    - resourceName: storageclasses
      allowed: ["standard"]
      default: standard
  grantPolicies:
    # A library ClusterResourceGrantPolicy without selectors.
    - corporate-issuers
```

For each source, the controller creates a service policy named `template-<template>-<source>` (for `spec.resources` — `template-<template>-inline`). A template that references a policy named `inline` is rejected, and so is a template whose service policy names would match those of another template. A reference to a non-existent policy or to a policy with either selector is rejected when the template is created or updated. A service policy selects the namespaces of the template’s projects with `namespaceSelector` by their `projects.deckhouse.io/project-template` label. The label is set when the template renders, so after a project switches to another template, the rules of the old template apply until the new one has rendered.

Granting cluster-wide resources to one project

A project can be given cluster-wide resources of its own, on top of the ones its template grants, in `spec.grants` of the Project. The field has the format of the template fields: `resources` with the rules themselves and `grantPolicies` with the names of library ClusterResourceGrantPolicy objects.

```
apiVersion: deckhouse.io/v1alpha3
kind: Project
metadata:
  name: my-project
spec:
  projectTemplateName: my-template
  grants:
    resources:
      - resourceName: storageclasses
        allowed: ["fast"]
    grantPolicies:
      - corporate-issuers
```

For each source, the controller creates a service policy named `project-<PROJECT_NAME>-<SOURCE>` (for `resources`, `project-<PROJECT_NAME>-inline`), owned by the project and selecting it with `projectSelector` by its `projects.deckhouse.io/project` label, so the policy applies to every namespace of the project, the ones created later included. Deleting the project deletes its service policies. The rules of the project and of its template combine as the rules of any two policies do: the allowed resources add up, and a resource that one of them denies stays unavailable.

The field has these rules:

- Changing `spec.grants` needs the right to create ClusterResourceGrantPolicy objects, the same right as creating such a policy directly. A user who may only change a project cannot grant it cluster-wide resources.
- The references and the names are checked like those of a template. A reference to a policy named `inline`, to a missing policy or to a policy with a selector is rejected. An empty or invalid `allowedSelector` or `deniedSelector` in `resources` is rejected. A `spec.grants` whose service policy names would match a policy that is not the project's, or the service policies of another project, is rejected, and so is a `spec.grants` whose service policy name is not a valid object name, which a very long name of a referenced policy causes.
- If a service policy cannot be applied, or a referenced policy is deleted later, the controller records a Warning event on the Project. The other sources stay applied, and a source removed from `spec.grants` is revoked all the same. A policy that is created again is picked up without waiting.
- The names that start with `project-` are reserved for the service policies, and a service policy is changed only through `spec.grants` of its project.
- `deckhouse.io/v1alpha2` has no place for the field, so a change of a project that has `spec.grants` through that version, a write of its status included, is rejected: it would erase the field. Change such a project through `deckhouse.io/v1alpha3`.

For project users

Project users can view the cluster-wide resources available to them and check the values used by default.

Viewing available cluster-wide resources

An AvailableClusterResource is automatically created in each project namespace for every registered resource. These resources show which cluster-wide resources are available to the project and which resource is used by default.

To view all cluster-wide resources available to a project, run the following command:

```
d8 k get available -n <PROJECT_NAME>
```

Example output:

```
NAME               KIND          DEFAULT      AVAILABLE   AGE
storageclasses     StorageClass  fast-ssd     2           5m
clusterissuers     ClusterIssuer letsencrypt  2           5m
```

To view detailed information about resources of a specific type, such as StorageClass, run the following command:

```
d8 k get available storageclasses -n <PROJECT_NAME> -o yaml
```

Example output (abridged):

```
status:
  grantedResourceKind: StorageClass
  default: fast-ssd
  availableCount: 2
  available:
    - name: fast-ssd
      default: true
      fields:
        provisioner: rbd.csi.ceph.com
        reclaimPolicy: Delete
        volumeBindingMode: WaitForFirstConsumer
        allowVolumeExpansion: true
    - name: hdd
      fields:
        provisioner: local.csi.storage.deckhouse.io
        reclaimPolicy: Retain
        volumeBindingMode: WaitForFirstConsumer
```

The `status.available[].fields` field contains the values of some fields of each available resource, so that you can choose a resource without access to the resource itself. The fields to show are selected by the module that registers the resource, in the `spec.catalogFields` parameter of its GrantableClusterResourceDefinition. For StorageClass, these are `provisioner`, `reclaimPolicy`, `volumeBindingMode` and `allowVolumeExpansion`. A value keeps its JSON type (for example, `allowVolumeExpansion` is a boolean). A field whose value is missing in the resource is not shown: in the example above, `hdd` has no `allowVolumeExpansion`. A change in a cluster-wide resource usually appears in AvailableClusterResource within 30 seconds.

An AvailableClusterResource exists as long as the corresponding GrantableClusterResourceDefinition: when the GrantableClusterResourceDefinition is deleted, its AvailableClusterResource is deleted from the namespaces of all projects.

Rejection due to an unavailable cluster-wide resource

If the cluster-wide resource is not available to the project when an object is created, or when an update sets a new value, the request is rejected with the following message:

```
[multitenancy] <KIND> "<OBJECT_NAME>" references "<RESOURCE_NAME>" which is not available to project "
<PROJECT_NAME>". Ask the cluster administrator to grant it.
```

In this case, check the available resources using AvailableClusterResource. Use a resource from the list of available resources or ask the cluster administrator to grant access to the required resource.

Assigning default values automatically

For some cluster-wide resources, the administrator can configure a default value. If the corresponding value is not specified when an object is created, it is assigned automatically.

For example, if `fast-ssd` is configured as the default StorageClass, the `fast-ssd` value can be automatically assigned to `.spec.storageClassName` when a PersistentVolumeClaim is created without this field.

You can specify a value explicitly by selecting any resource available to the project from the corresponding AvailableClusterResource.

Projects without a default value

AvailableClusterResource reports the project default in `status.default` and marks it among the names in `status.available` with `default: true`. Both are absent in the following cases:

- the grant policy sets no `default`, and the registration has no `defaultFrom`;
- the resource is granted as a list of values rather than as objects of the cluster, and `defaultFrom` does not apply to it;
- `defaultFrom` is set, but no object has its annotation set to `true`. The `DefaultFromResolved` condition of the GrantableClusterResourceDefinition shows this with the reason `NoDefault`;
- a default is set, but the project may not use it, for example the default StorageClass of the cluster is not among the names granted to the project, or the default is a reserved resource that no policy

allows for the project.

Then the field stays empty, and the object is created or rejected by the usual rules.

The cluster-wide default that `defaultFrom` finds is chosen the way Kubernetes chooses the default StorageClass. An object is the default if one of the annotations of `defaultFrom` is set to exactly `true`, so `false` and `True` do not count. For StorageClass, both `storageclass.kubernetes.io/is-default-class` and the older `storageclass.beta.kubernetes.io/is-default-class` count. If several objects are marked, the most recently created one is used. The `DefaultFromResolved` condition of the GrantableClusterResourceDefinition lists the marked objects and names the one used:

```
d8 k get grantableclusterresourcedefinition storageclasses -o jsonpath='{.status.conditions[?
(@.type=="DefaultFromResolved")].message}'
```

For module developers

Configuring validation of a cluster-wide resource reference

If a module resource contains a field that references a cluster-wide resource already registered using GrantableClusterResourceDefinition, create a GrantableClusterResourceReference. It defines which resources and fields use the cluster-wide resource and allows you to configure availability checks and automatic default value assignment.

For example, to configure validation of the StorageClass specified in the `.spec.storageClassName` field of a PostgresDatabase resource, add the following object to the module's Helm chart:

```
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceReference
metadata:
  name: postgresdatabases-storageclasses
  labels:
    heritage: deckhouse
    module: postgres
spec:
  grantableClusterResourceName: storageclasses
  rule:
    apiGroups: ["postgres.example.com"]
    apiVersions: ["*"]
    resources: ["postgresdatabases"]
  fieldPaths:
    - path: $.spec.storageClassName
      defaulting: Coerce
```

In this example, `storageclasses` is the name of an existing GrantableClusterResourceDefinition, while `Coerce` allows the default StorageClass available to the project to be assigned when a PostgresDatabase is created if the value is missing or unavailable to the project.

If the resources listed in `rule` store the reference at different paths, limit `fieldPaths` items to specific resources using the `resources` parameter. For example, a module introduces the MyBatch and MyBatchSchedule resources that store the name of a PriorityClass (registered by Deckhouse Platform as `priorityclasses`) at different paths. In this case, use the following example:

```
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceReference
metadata:
  name: mybatches-priorityclasses
  labels:
    heritage: deckhouse
    module: my-module
spec:
  grantableClusterResourceName: priorityclasses
  rule:
    apiGroups: ["batch.example.com"]
    apiVersions: ["*"]
    resources: ["mybatches", "mybatchschedules"]
  fieldPaths:
    - resources: ["mybatches"]
      path: $.spec.workerPriorityClassName
    - resources: ["mybatchschedules"]
      path: $.spec.batchTemplate.spec.workerPriorityClassName
```

For each object, the most specific of the matching `fieldPaths` items is used: an item limited by `resources` takes precedence over items limited only by `apiGroups` or `apiVersions` and over an item without limits, which applies to all other resources from `rule`. `*` in `resources` does not limit the item: it has the same effect as omitting the parameter.

An item whose `match` condition does not hold for the object is skipped, and of equally limited items, an item with `match` takes precedence over an item without it. So several items with the same `path` can handle different objects of one resource, each with its own default assignment mode: the `storageclasses-pvc` reference checks the StorageClass of a PersistentVolumeClaim whose `spec.dataSource` or `spec.dataSourceRef` is a VolumeSnapshot or a PersistentVolumeClaim or that has the `volume.beta.kubernetes.io/storage-class` annotation in the `None` mode and of any other PersistentVolumeClaim in the `Coerce` mode. The text of the `message` parameter is appended to the rejection of an object for that item.

If the reference is stored in the elements of a list of templates, an element can leave the field out, for example a volume claim template of a StatefulSet without `storageClassName`. Such an element gets the cluster-wide default, for example the default StorageClass of the cluster, which `path` does not see. To check it, set the `missingPath` parameter of the item to a JSONPath that selects the elements without the reference, for example `$.spec.volumeClaimTemplates[?!@.spec.storageClassName]`.

If the GrantableClusterResourceDefinition has `defaultFrom`, an object in which `missingPath` selects anything is rejected only if the cluster-wide default that `defaultFrom` finds is not available to the project, and is admitted if there is no such default. Without `defaultFrom`, such an object is rejected in a project to which a ClusterResourceGrantPolicy with an entry for the cluster-wide resource applies, and admitted in other projects. Default assignment cannot fill in an element of a list, so use `missingPath` with `defaulting: None`.

For descriptions of GrantableClusterResourceReference parameters, default assignment modes, `match` conditions, and configuration for resources with multiple API versions, refer to the resource description.

Registering a new cluster-wide resource

To add access management for a new cluster-wide resource:

1. Create a GrantableClusterResourceDefinition to register the cluster-wide resource with the access management mechanism.
2. Create one or more GrantableClusterResourceReference objects to define the fields that reference it.

   For example, to register the MyClusterResource resource, add the following object to the module’s Helm chart:

```
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceDefinition
metadata:
  name: myclusterresources
  labels:
    heritage: deckhouse
    module: my-module
spec:
  grantedResource:
    apiGroup: my.example.com
    kind: MyClusterResource
  enforcement: Managed
  defaultAvailability: All
  excluded:
    - matchExpressions:
        - key: my.example.com/internal
          operator: Exists
```

In this example, the registered resources are available to projects by default. Resources with the `my.example.com/internal` label are excluded from the available resources.

To keep some resources unavailable by default but let an administrator grant them to individual projects, list them in `reserved` instead of `excluded`. An administrator grants such a resource as described in “Granting a reserved resource”.

Only a cluster-scoped resource can be specified in `grantedResource`. A definition of a namespaced resource (for example, Secret) is refused when it is created or updated. If such a definition is stored anyway, it is not processed: no project gets its AvailableClusterResource, and the `GrantedResourceValid` condition is `False` (read more in section “Checking resource registration status”). A resource whose CRD is not installed yet is accepted: its scope is checked once the resource is served.

For descriptions of GrantableClusterResourceDefinition parameters and available access management modes, refer to the resource description.

3. After registering the cluster-wide resource, configure references to it using GrantableClusterResourceReference as described in “Configuring validation of a cluster-wide resource reference”.

Showing resource fields to projects

Project users see the available cluster-wide resources in AvailableClusterResource (read more in section “Viewing available cluster-wide resources”), but usually cannot read the resources themselves. To help them choose, list the fields to show in `spec.catalogFields` of the GrantableClusterResourceDefinition. The controller copies the values of these fields from every available resource into `status.available[].fields` of AvailableClusterResource.

For example:

```
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceDefinition
metadata:
  name: myclusterresources
spec:
  grantedResource:
    apiGroup: my.example.com
    kind: MyClusterResource
  catalogFields:
    - name: tier
      path: $.spec.tier
    - name: region
      path: $.metadata.labels['my.example.com/region']
```

Rules:

- `name` is the key in `fields`, in lowerCamelCase; `path` is an RFC 9535 JSONPath singular query that selects one value: member names and array indexes only, without wildcards (`*`), descendant segments (`..`), slices or filters.
- Paths that read `metadata.managedFields` or the `kubectl.kubernetes.io/last-applied-configuration` annotation, or a parent that contains them (for example, `$.metadata`), are not allowed: these fields hold a copy of the whole resource.
- At most 10 fields can be listed.
- A value longer than 1024 bytes in JSON serialization is not shown; a field whose value is missing in the resource is not shown either.
- If the `status.available` list of an AvailableClusterResource with the fields of all resources would take more than 512 KiB in JSON serialization, no fields are shown in it; the names and the default stay.
- `catalogFields` works only for definitions with `grantedResource`, which must be cluster-scoped.
- The values are visible to all users of every project the resource is available to. List only non-secret data.

Invalid paths and `catalogFields` without `grantedResource` are refused when the GrantableClusterResourceDefinition is created or updated, and are reported in its `CatalogFieldsValid` condition. Values left out for their size depend on the resources and the project, so they are reported by `CatalogFieldsSkipped` warning events on the GrantableClusterResourceDefinition:

```
d8 k describe grantableclusterresourcedefinition myclusterresources
```

Using x-deckhouse-grantable-resource in Deckhouse Platform application settings

To manage access to cluster-wide resources in Deckhouse Platform application settings, use the `x-deckhouse-grantable-resource` OpenAPI extension. In this case, deckhouse-controller automatically checks the availability of the specified resource and assigns the default value when necessary. You do not need to create GrantableClusterResourceReference manually.

For a description of the extension and usage examples, refer to “Application settings”.

Checking resource registration status

You can check the status of GrantableClusterResourceDefinition and its associated GrantableClusterResourceReference resources in the `status` field:

- `GrantableClusterResourceDefinition.status.references`: Contains a list of associated GrantableClusterResourceReference resources and information about the resources to which they apply.
- `GrantableClusterResourceReference.status.bound`: Indicates whether the corresponding GrantableClusterResourceDefinition was found.
- `GrantableClusterResourceReference.status.conditions[Bound]`: `True` with the reason `Resolved` if the definition was found, `False` with the reason `UnknownResource` if it is missing. `UnknownResource` can indicate an incorrect GrantableClusterResourceDefinition name or a missing registration.
- `GrantableClusterResourceReference.status.conditions[FieldPathsValid]`: Whether `spec.fieldPaths` pass the checks applied when the object is created or updated (reasons `Valid`, `InvalidFieldPaths`; the message lists all problems). `False` means the object was stored bypassing these checks; the invalid paths are skipped during availability checks.
- `GrantableClusterResourceDefinition.status.conditions[GrantedResourceValid]`: Whether `spec.grantedResource` can be processed. `True` with the reason `ClusterScoped` for a cluster-scoped resource or `ValueBacked` for a definition without `grantedResource`; `False` with the reason `Namespaced` for a namespaced resource, which is not processed; `Unknown` with the reason `KindNotServed` while the resource is not served (for example, its CRD is not installed yet) or `MappingFailed` if its scope could not be determined.
- `GrantableClusterResourceDefinition.status.conditions[CatalogFieldsValid]`: Whether `spec.catalogFields` pass the checks applied when the object is created or updated (reasons `Valid`, `InvalidCatalogFields`; the message lists all problems). A definition without `catalogFields` has the condition `True`. Values left out for their size are reported by `CatalogFieldsSkipped` events (read more in section “Showing resource fields to projects”).

For example:

```
d8 k get grantableclusterresourcedefinition myclusterresources -o jsonpath='{range
.status.conditions[*]}{.type}={.status} ({.reason}): {.message}{"\n"}{end}'
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The multitenancy-manager module: FAQ

> ⚠ The module lifecycle stage: Preview

Managing access to cluster-wide resources

What should I do if a PersistentVolumeClaim is rejected with an “is not available to project” error?

The StorageClass specified in `spec.storageClassName` is unavailable to the project. The webhook rejects such a request with a message similar to the following:

```
[multitenancy] PersistentVolumeClaim "<OBJECT_NAME>" references "<RESOURCE_NAME>" which is not available
to project "<PROJECT_NAME>". Ask the cluster administrator to grant it.
```

To view the available StorageClasses, run the following command:

```
d8 k get available storageclasses -n <PROJECT_NAME> -o yaml
```

If the required StorageClass is not in the list, use an available one or ask the cluster administrator to add it to the ClusterResourceGrantPolicy.

You can check the availability of other cluster-wide resources, such as ClusterIssuer, ClusterRole, and LoadBalancerClass, in the same way using the corresponding AvailableClusterResource.

How can I view access policies applied to a project?

A ClusterResourceGrantPolicy applies to a namespace of a project when every selector it sets matches: `spec.projectSelector` is matched against the labels of the Project, `spec.namespaceSelector` against the labels of the namespace. A policy with neither selector is a library policy and applies only through `spec.grantPolicies` of a ProjectTemplate. For details, refer to “How a policy selects namespaces”.

To view all policies and their selectors, run the following command:

```
d8 k get clusterresourcegrantpolicies -o yaml
```

You can view the resulting list of cluster-wide resources available to a specific project using AvailableClusterResource:

```
d8 k get available -n <PROJECT_NAME>
```

What happens to existing objects when access is restricted?

Existing objects continue to use their previously configured values. When an object is updated, only new values are checked, so changing an access policy does not disrupt existing objects. The same holds when access management is enabled on an existing cluster: existing objects do not need to be changed before access policies are created.

If an existing object needs to use a different cluster-wide resource, explicitly change the corresponding field to a value available to the project.

If an existing object uses a cluster-wide resource that is no longer available to the project, the `ClusterResourceGrantPolicyViolation` alert is triggered.

What does the ClusterResourceGrantPolicyViolation alert mean?

The alert indicates that one or more existing objects in a project use a cluster-wide resource that is no longer available to the project under the current policies.

Such objects continue to operate. To resolve the discrepancy, grant the project access to the cluster-wide resource in use or modify the object to use an available resource.

Detailed information about violations is available on the Grafana dashboard under “Security” → “Cluster Resource Grant Violations”. The `d8_cluster_objects_grant_violated` metric is used for monitoring; the module controller exports it, applying the same availability rules as the admission webhook.

How can I allow all StorageClasses except specific ones?

To deny specific StorageClasses while keeping the others available, use `denied` or `deniedSelector`. Deny rules take precedence over allow rules.

For a configuration example, refer to “Denying individual resources”.

What happens if there is no ClusterResourceGrantPolicy?

If no ClusterResourceGrantPolicy is configured for a resource, its availability is determined by its registration. The resource is available to all projects if `defaultAvailability: All` (the default value) is set in GrantableClusterResourceDefinition and the resource does not match the `excluded` or `reserved` filters.

For example, the `clusterroles` definition provided by Deckhouse Platform excludes every ClusterRole without the `rbac.deckhouse.io/delegatable: "true"` label. Therefore, such roles are unavailable in RoleBinding even when no policies are configured.

To restrict access, create a ClusterResourceGrantPolicy and specify the projects and the cluster-wide resources available to them.

I created a ClusterResourceGrantPolicy, but nothing changed — why?

Check the following:

1. **Selector match**. Make sure the policy’s `spec.projectSelector` matches the labels of the Project, and its `spec.namespaceSelector` the labels of the namespace:

   Get the project and its labels:

   ```
   d8 k get project <PROJECT_NAME> --show-labels
   ```

   Filter the namespaces by the `projects.deckhouse.io/project=<PROJECT_NAME>` label and show their labels:

   ```
   d8 k get ns -l projects.deckhouse.io/project=<PROJECT_NAME> --show-labels
   ```

   In a `v1alpha1` manifest, `spec.projectSelector` is what `v1alpha2` calls `spec.namespaceSelector`. Also check the conditions in the policy status. The `SelectorsValid` condition is `False` when a selector is invalid. The `NewNamespacesCovered` condition is `False` when a restriction does not cover a new additional namespace of the project. The `AllowedEffective` condition is `False` when the GrantableClusterResourceDefinition excludes a name listed in `allowed`, for example a ClusterRole without the `rbac.deckhouse.io/delegatable: "true"` label. The message names it (see Restricting ClusterRoles for a project).

2. **Resources available to the project**. Check whether AvailableClusterResource reflects the expected policy settings:

   ```
   d8 k get available <RESOURCE_NAME> -n <PROJECT_NAME> -o yaml
   ```

3. **Cluster-wide resource registration**. Make sure a corresponding GrantableClusterResourceDefinition exists for the `resourceName` value.

4. **Reference registration**. If a specific field is expected to be checked, make sure it is registered using GrantableClusterResourceReference and that the reference is successfully associated with the corresponding GrantableClusterResourceDefinition.

For information about registration status, refer to “Checking resource registration status”.

How does cluster-wide resource access management interact with RBAC?

These mechanisms operate independently. RBAC determines *who can perform* operations on an object, while cluster-wide resource access management determines *which cluster-wide resources* that object can use.

For example, to create a PersistentVolumeClaim, a user must have the appropriate RBAC permissions, and the StorageClass specified in `.spec.storageClassName` must be available to the project.

What happens if the multitenancy-manager module is disabled?

After the module is disabled, cluster-wide resource availability checks and automatic default value assignment are no longer performed. Existing objects remain unchanged.

Fields that use the `x-deckhouse-grantable-resource` extension in Deckhouse Platform application settings are no longer checked either.

The `d8:dict` role of the `user-authz` module continues to work.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The namespace-configurator module

> **i** The module lifecycle stage: General Availability

The `namespace-configurator` module allows to assign annotations and labels to namespaces automatically.

It facilitates to enable new namespaces to monitoring system by adding `extended-monitoring.deckhouse.io/enabled=true` label.

How does it work?

This module monitors the namespaces and configuration:

- It assigns labels and annotations from the configuration to all namespaces that match the `includeNames` pattern and do not match the `excludeNames` pattern.
- Namespaces labeled `heritage` with one of the values `upmeter`, `deckhouse` or `multitenancy-manager` are ignored;
- When changing the module configuration, namespace labels and annotations will be reassigned according to the configuration.

What do I need to configure?

All you need to do is to specify list of desired labels and annotations and matching patterns for namespaces in the module configuration.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

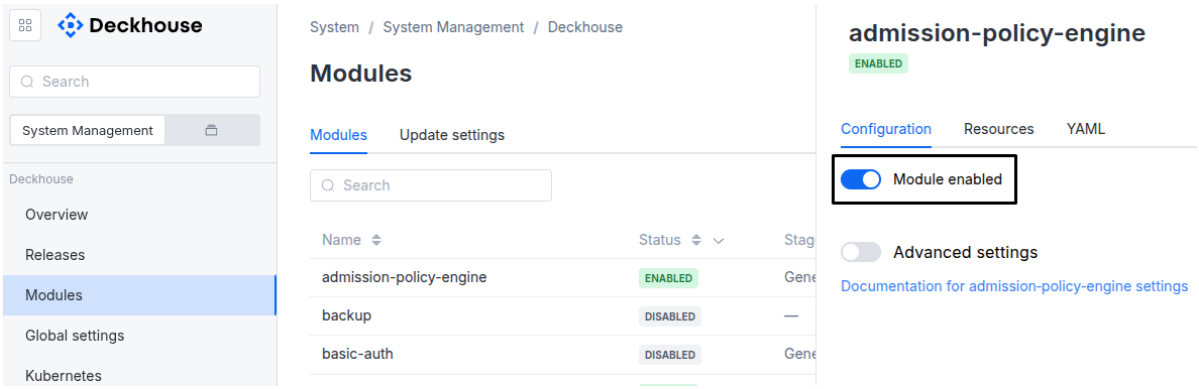
> The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `namespace-configurator` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable namespace-configurator
```

- **Using ModuleConfig `namespace-configurator`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `namespace-configurator` (create it if necessary);

  Example of a manifest to enable module `namespace-configurator`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  enabled: true
```

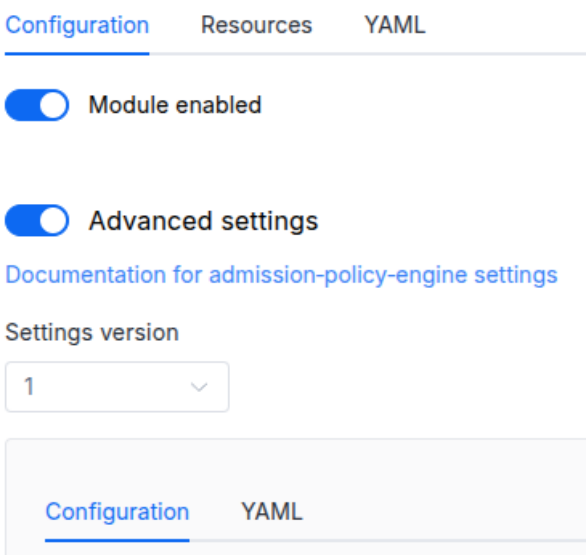
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `namespace-configurator` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `namespace-configurator` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `namespace-configurator`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `namespace-configurator` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc namespace-configurator
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `namespace-configurator` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `namespace-configurator`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: namespace-configurator
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.configurations**
    array of objects
    - **settings.configurations.annotations**
      object

      A list of annotations in the `key: "value"` format.

      Example:

```
annotations:
  example: 'true'
```

- **settings.configurations.excludeNames**
  array of strings

  A list of namespace patterns to exclude.

  Default: `[]`

  Example:

  ```
  excludeNames:
  - infra-test
  ```

- **settings.configurations.includeNames**
  array of strings

  Required value

  A list of namespace patterns to include.

  Default: `[]`

  Example:

  ```
  includeNames:
  - prod-.*
  - infra-.*
  ```

- **settings.configurations.labels**
  object

  A list of labels in the `key: "value"` format.

  Example:

  ```
  labels:
    istio-injection: 'true'
  ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Example

Add the label `extended-monitoring.deckhouse.io/enabled=true` and the annotation `foo=bar` to every namespace starting with `prod-` and `infra-`, except `infra-test`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
    - annotations:
        foo: bar
      labels:
        extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
      - "^prod"
      - "^infra"
      excludeNames:
      - "infra-test"
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The network-gateway module

> The module lifecycle stage: General Availability

This module creates a network gateway using Kubernetes nodes. It contains a DHCP server and a SNAT manager.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

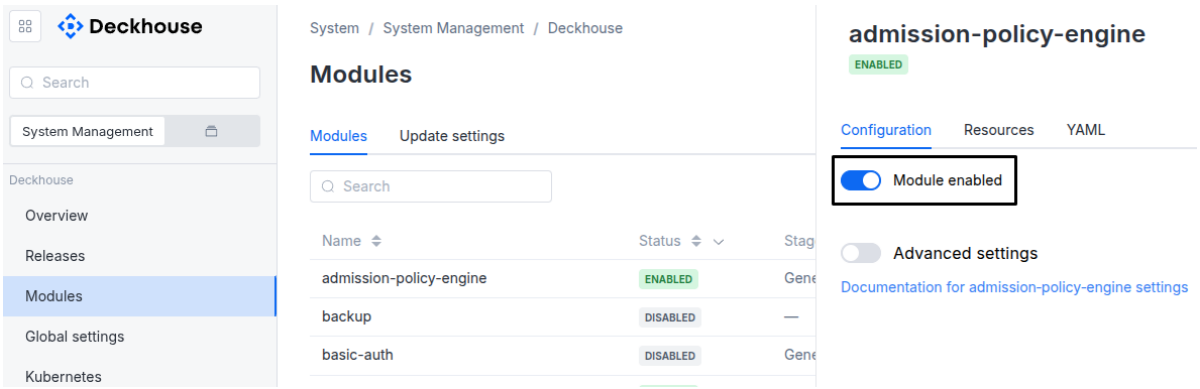
> The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-gateway` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable network-gateway
  ```

- **Using ModuleConfig `network-gateway`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `network-gateway` (create it if necessary);

  Example of a manifest to enable module `network-gateway`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-gateway
  spec:
    enabled: true
  ```

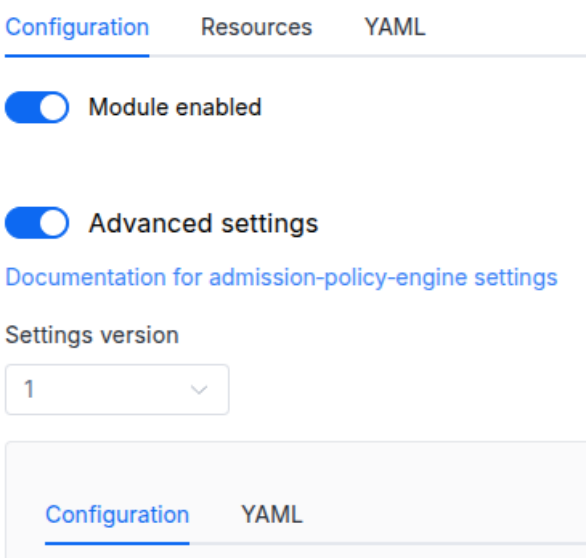
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-gateway` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `network-gateway` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the

module `network-gateway`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `network-gateway` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc network-gateway
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `network-gateway` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `network-gateway`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-gateway
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableDHCP**
    boolean

    Disables the DHCP server.

    Default: `false`
  - **settings.dns**
    object

    Settings to pass to clients via DHCP.

    - **settings.dns.search**
      array of strings

      List of search domains.

      Example:

      ```
      search:
      - office.example.com
      - srv.example.com
      ```
    - **settings.dns.servers**
      array of strings

      List of DNS servers.

      Example:

```
servers:
- 4.2.2.2
- 8.8.8.8
```

- **settings.nodeSelector**
  object

  Required value

  Selects nodes that will be used to configure iptables rules and to run the DHCP server.

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes. Instance Pods inherit this field as is.

  Example:

  ```
  nodeSelector:
    type: network-gateway
  ```

- **settings.publicAddress**
  string

  Required value

  Replaces the src of the packets outgoing from the LAN.

  Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

  Example:

  ```
  publicAddress: 10.220.203.240
  ```

- **settings.storageClass**
  string

  The name of the StorageClass to use for storing the DHCP lease.

  If omitted, the StorageClass of the existing PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  *Warning.** Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  If `false` is specified, emptyDir will be forced to be used.

  Dnsmasq (underlies our DHCP server) has its own mechanisms for protecting against the duplication of IP addresses if the lease database is lost (but it is better not to lose it).

- **settings.subnet**
  string

  Required value

  The address of a local subnet that gateway serves.

  The DHCP options to pass to clients are generated based on this address:

  - Address pool — numbers starting with 50 and up to the last one.
  - Router — the subnet's first address.

  Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}(\/([0-9]|[1-2][0-9]|3[0-2]))$`

  Example:

  ```
  subnet: 192.168.42.0/24
  ```

- **settings.tolerations**
  array of objects

  Tolerations for DHCP Pods and iptables managers.

The same as in the Pods' `spec.tolerations` parameter in Kubernetes. The instance's Pods inherit this field as is.

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The network-policy-engine module

> The module lifecycle stage: General Availability

> ⚠ Do not use the module if the cilium module is enabled because it already has network policy management.

This module manages network policies.

Deckhouse implements a conservative approach to organizing the network based on elementary network backends, such as "pure" CNI or flannel in the `host-gw` mode. This approach is simple and reliable, and has proven itself to be the best.

The NetworkPolicy implementation in Deckhouse is also simple and reliable. It is based on `kube-router` in the Network Policy Controller mode (`--run-firewall`). In this case, `kube-router` transforms NetworkPolicy policies into `iptables` rules that work in any installation regardless of the cloud or the CNI used.

The `network-policy-engine` module deploys a `d8-system` DaemonSet in the namespace with kube-router in the Network Policy Controller mode. As a result, the Kubernetes cluster fully supports Network Policies.

The following policy description formats are supported:

- `networking.k8s.io/NetworkPolicy API`;
- network policy V1/GA semantics;
- network policy beta semantics.

For the NetworkPolicy model and ready-to-use recipes, see the Network policies section of the documentation. Additional example recipes are available in an external repository.

Third-party components

List of third-party software used in the `network-policy-engine` module:

- Kube-router 2.0.1

  License: Apache License 2.0

  A turnkey solution for Kubernetes networking with aim to provide operational simplicity and high performance.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

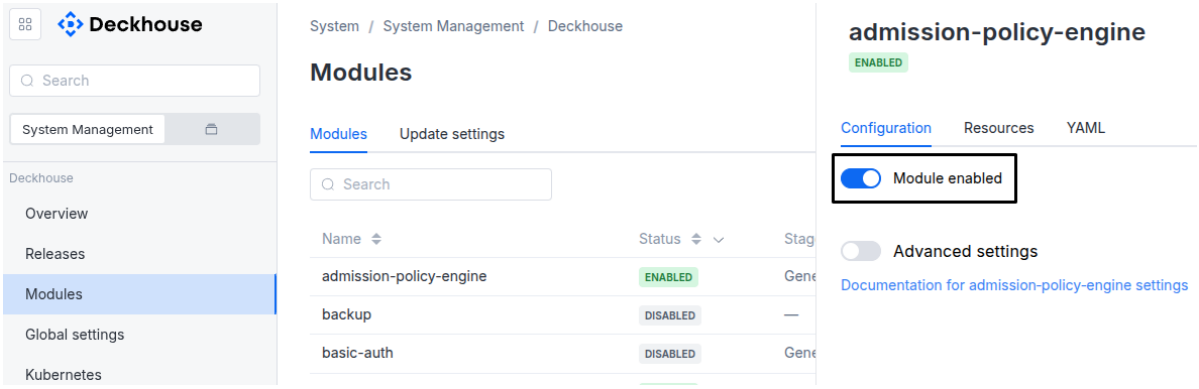
> The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-policy-engine` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:

- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable network-policy-engine
  ```

- **Using ModuleConfig `network-policy-engine`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `network-policy-engine` (create it if necessary);

  Example of a manifest to enable module `network-policy-engine`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-policy-engine
  spec:
    enabled: true
  ```

How to configure the module...

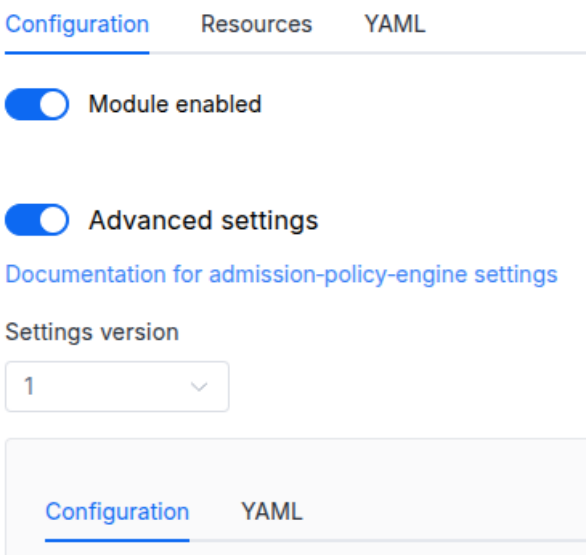
You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-policy-engine` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab,

excluding the settings section. Save the changes.

Example:



You can also edit the ModuleConfig object `network-policy-engine` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `network-policy-engine`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `network-policy-engine` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc network-policy-engine
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `network-policy-engine` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `network-policy-engine`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-policy-engine
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

Examples of network policies

Deny external traffic to Pods in the namespace, but allow traffic to external resources from within the namespace

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: delete
spec:
  podSelector: {}
  egress:
  - {}
  ingress:
  - from:
    - podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

More examples are available in this repository.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The node-local-dns module

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

Module Description

This module deploys a caching DNS server on each cluster node and exports data to `Prometheus` for easy analysis of the cluster's DNS performance using the Grafana board.

The module runs the standard `CoreDNS` server (as part of a DaemonSet) on all nodes supplemented by the algorithm for configuring network and iptables rules.

Purpose

The regular DNS usage in Kubernetes poses some problems that may lead to unnecessary degradation of the service's performance:

- No request caching (in Linux, there is no caching right out-of-the-box; thus, Pods also lack it out-of-the-box).
- All the container-originated DNS requests result in a network request to the cluster DNS. Thus, even requests to resources within the same node result in network requests.
- The Pod request is first resolved in the cluster DNS zones and only then channeled to external DNS servers. For example, the request to `ya.ru` is first resolved in cluster zones such as `cluster.local`, `svc.cluster.local`, `<namespace>.svc.cluster.local`, receives negative responses, and gets appropriately resolved only on the "second" attempt.

Thus, even slight network delays may result in significant service performance degradation due to above problems.

One solution is to deploy a DNS server on each node (which is what this module does).

Outside requests (non-cached ones) are also first resolved in internal zones. Under high load (when a large number of requests are made to the same records simultaneously, as is often the case), such behavior can significantly improve the performance of DNS resolving.

Grafana dashboard

The `Kubernetes / DNS (node local)` dashboard displays:

- General graphs that allow you to evaluate the overall DNS performance.
- Graphs grouped by the node that let you analyze node data in more details if the general charts suggest that this node is unhealthy.
- Upstream-related graphs that allow you to evaluate the performance of the cluster DNS as well as the node servers specified in `/etc/resolv.conf`.

How does it work?

The module performs the following settings on each node:

- Configures an interface with the ClusterIP IP address of the `kube-dns` service.
- Starts a caching CoreDNS server that listens on this address.
- Adds a sophisticated iptables rule that uses a socket if it is open and ordinary Kubernetes magic for clusterIP if it is not.

And here is the rule itself that allows for an easy fallback:

```
-A PREROUTING -d <kube-dns IP address> -m socket --nowildcard -j NOTRACK
```

Aspects of CoreDNS configuration

Main features of the CoreDNS config:

- caching of all requests;
- forwarding all DNS requests to ClusterIP of the cluster DNS.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> **i** The module lifecycle stage: General Availability
>
> The module has requirements for installation

> **⚠**
> - The module works with all `CNIs`, but in order to work correctly with `cni-cilium`, a number of conditions must be met.
> - By default, when used together with the `cni-simple-bridge` or `cni-flannel` modules, the `node-local-dns` module **does not work** for requests from `hostNetwork`. In this case, all requests go to the `kube-dns` module. You can specify the address `169.254.20.10` in the pod configuration, but if `node-local-dns` module crashes, there will be no *fallback* to the `kube-dns` module.

The module has 5 alerts.

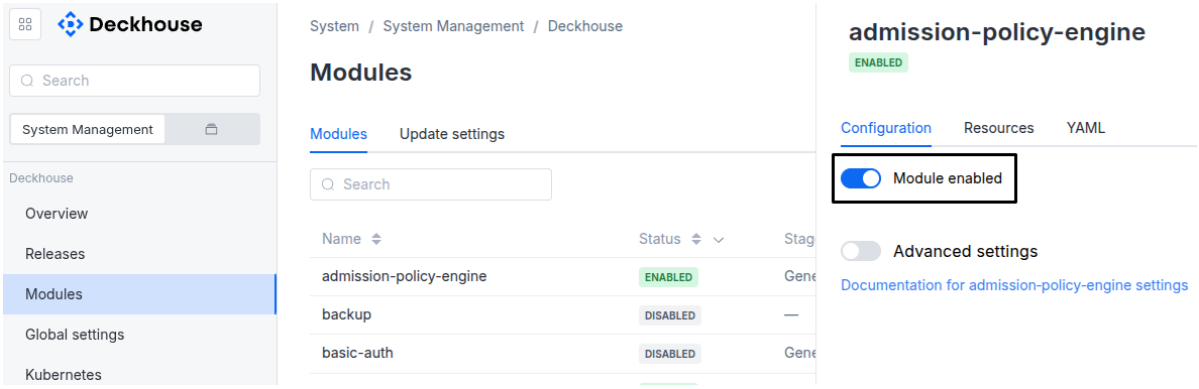
The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-local-dns` module and enable (or disable) the “Module enabled” toggle.

Save changes.

Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable node-local-dns
  ```

- **Using ModuleConfig `node-local-dns`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `node-local-dns` (create it if necessary);

  Example of a manifest to enable module `node-local-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: node-local-dns
  spec:
    enabled: true
  ```

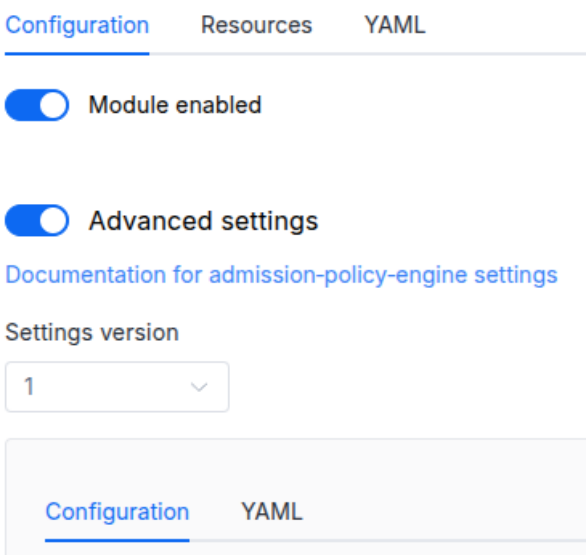
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-local-dns` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `node-local-dns` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `node-local-dns`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `node-local-dns` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc node-local-dns
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `node-local-dns` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `node-local-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: node-local-dns
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `kube-dns`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableIPv6**
    boolean

Disables DNS resolution over IPv6 by returning empty responses for `AAAA` queries and `NXDOMAIN` for reverse IPv6 lookup requests in the `ip6.arpa` zone.

Default: `false`

- **settings.enableLogs**
  boolean

  Enables query logging for node-local-dns instances.

  Default: `false`

- **settings.highAvailability**
  boolean

  Manual failover mode management. Within this module, the option affects the DNS records caching mechanism on nodes:

  - `true`: upon the expiration of the TTL for an individual DNS record, node-local-dns will continue to return the “stale” value immediately while simultaneously requesting a fresh value from the upstream kube-dns. Thus, in the event of issues with the nodes where kube-dns is running, applications will continue to receive responses to DNS queries with expired TTL, but without delay.
  - `false`: upon the expiration of the TTL for an individual DNS record, node-local-dns will continue to return the “stale” value, but only after querying the upstream kube-dns. Thus, in the event of issues with the nodes where kube-dns is running, applications will continue to receive DNS responses with expired TTL, but with potentially significant delay. This approach is suitable in highly dynamic environments where Services and Namespaces are frequently recreated.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

  Example:

  ```
  highAvailability: true
  ```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

The module lifecycle stage: General Availability

The module has requirements for installation

An example of configuring a custom DNS for a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx
```

Here you can learn more about DNS configuring.

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  version: 1
  enabled: true
  settings:
    enableLogs: true
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Node-manager module

> The module lifecycle stage: General Availability

The `node-manager` module manages nodes in a Kubernetes cluster deployed with Deckhouse Platform. It configures and updates node software, supports cloud and static nodes, and manages Linux users.

Main features

The module provides the following features:

1. Managing multiple nodes as a related group (**NodeGroup**):
   - The ability to define metadata that are inherited by all nodes in the group.
   - Monitoring of a group as a single entity (grouping nodes on graphs by groups, grouping alerts about node unavailability, alerts about the unavailability of N or N% of nodes in a group).
2. **Chaos monkey** — the systemic termination of nodes. This feature tests the resilience of cluster elements and running applications.
3. Delaying the reboot or shutdown of a node while critical pods are running on it.
4. Installing, updating, and configuring the node software (containerd, kubelet, etc.), connecting the node to the cluster:
   - Installing operating system (see the list of supported OS) regardless of the infrastructure used (any cloud/any hardware).
   - The operating system’s basic setup (disabling auto-update, installing the necessary packages, configuring logging parameters, etc.).
   - Configuring kubernetes-api-proxy (and the system for automatically updating the list of upstreams) to balance node (kubelet) requests over API servers.
   - Installing and configuring CRI containerd and Kubernetes, adding the node to the cluster.
   - Managing node updates and their downtime (disruptions):
     - Automatic determination of a valid minor Kubernetes version for a node group based on its settings (the kubernetesVersion parameter specified for a group), the default version for the whole cluster, and the current control-plane version (no nodes can be updated ahead of the control-plane update).
     - Only one node of a group can be updated at a time, and only if all the nodes in the group are available.
     - There are two types of node updates:
       - regular updates – always performed automatically;
       - disruption-involving updates (such as updating the kernel, switching docker versions, major change of the kubelet versions, etc.) – you can choose manual or automatic mode. If automatic disruptive updates are enabled, the node is drained before the update (this functionality can be disabled).
   - Monitoring the status and progress of the update.
5. Cluster scaling.

   - Autoscaling.

Available with supported cloud providers (learn more); not available for static nodes. Cloud providers support automatic creation or deletion of virtual machines, joining them to or disjoining them from the cluster.

- Maintaining the desired number of nodes in a group.

  Available for both cloud providers and static nodes (when using Cluster API Provider Static).

6. Managing Linux users on nodes.

Nodes are managed through the NodeGroup resource, and each node group performs specific tasks. Below are examples of node groups pooled by their tasks performed:

- a group of master nodes;
- a group of nodes for routing HTTP(s) traffic (front nodes);
- a group of monitoring-related nodes;
- a group of application nodes (the so-called worker nodes), etc.

The nodes belonging to the group have common parameters and are configured automatically according to the group's parameters. Deckhouse scales groups by adding, excluding, and updating their nodes. Both Static nodes deployed in the cloud and static (bare-metal or virtual machine) nodes can be combined into a single group. It paves the way for the hybrid clusters, in which you can scale nodes on physical servers via cloud nodes.

Operations on the cloud infrastructure are performed by means provided by supported cloud providers. If there is no support for the desired cloud platform, you can use its resources as static nodes.

The static nodes (e.g., bare metal servers) are handled by the Cluster API Provider Static (CAPS).

The following Managed Kubernetes services are supported (note that some service functionality may be unavailable):

- Google Kubernetes Engine (GKE);
- Elastic Kubernetes Service (EKS).

Node types

The following node types that can be worked with within a node group (resource NodeGroup) are supported:

- `CloudEphemeral` — the nodes are automatically ordered, created, and deleted in the configured cloud provider or on physical servers using the `infra-provider-baremetal` module.
- `CloudPermanent` — the nodes differ in that their configuration is not taken from the custom resource nodeGroup, but from a special resource `<PROVIDER>ClusterConfiguration` (for example, AWSClusterConfiguration for AWS). Also, an important difference is that to apply node configuration, you need to run `dhctl converge` (by running Deckhouse installer). An example of a CloudPermanent node of a cloud cluster is a cluster master node.
- `CloudStatic` — a static node (created manually) hosted in the cloud integrated with one of the cloud providers. This node has the CSI running, and it is managed by the cloud-controller-manager. The `Node` object automatically gets the information about the cloud zone and region. Also, if a node gets deleted from the cloud, its corresponding Node object will be deleted in a cluster.
- `Static` — a static node hosted on a bare metal or virtual machine. In the case of a cloud environment, the `cloud-controller-manager` does not manage the node even if one of the cloud providers is enabled. Learn more about working with static nodes...

> **i** For Deckhouse Platform virtualization, the DVPClusterConfiguration resource is not used. The CloudPermanent node parameters are set in the NodeGroup, DVPInstanceClass, and `cloud-provider-dvp` ModuleConfig resources.

Node grouping and group management

Grouping and managing nodes as a related group mean that all nodes in the group will have the same metadata derived from the `NodeGroup` custom resource.

The following monitoring patterns are available for node groups:

- with grouping of node parameters on the graphs for the group;
- with grouping of alerts about node unavailability;
- with alerts about unavailability of N nodes or N% of nodes in the group, etc.

Delaying node reboot or shutdown while critical pods are running

The `d8-shutdown-inhibitor` service in `node-manager` delays node shutdown or reboot if stateful applications or virtual machines (VMs) are running on the node. The delay is triggered if pods with the label `pod.deckhouse.io/inhibit-node-shutdown` are running on the node. Deckhouse Platform delays the shutdown or reboot of the node so that the user can migrate critical pods or terminate them in a controlled manner. This is useful, for example, for stateful workloads and other scenarios where data loss must be avoided.

How it works and key features

The `d8-shutdown-inhibitor` service prevents a node from shutting down as long as there are pods on it with the `Running` status that are labeled `pod.deckhouse.io/inhibit-node-shutdown`. The `GracefulShutdownPostpone` condition is used for this blocking. Additionally, the node is cordoned off during the blocking period to prevent the scheduler from scheduling new pods on it. The kubelet reads the node's `GracefulShutdownPostpone` condition and does not begin evicting pods marked with this label as long as the condition's `status` is `True` and `reason` is `PodsWithLabelAreRunningOnNode`. If there are pods on the node with the label `pod.deckhouse.io/inhibit-node-shutdown`, a corresponding message is displayed in the node console (wall).

Delaying mechanism features in Deckhouse Platform

The delay in restarting or shutting down a node while critical pods are running on it has the following features in Deckhouse Platform:

- A label must be explicitly set on each pod that is to block shutdown. If pods are created via a Deployment, the label is specified in the pod template.
- This mechanism works only with a modified kubelet version included in Deckhouse Platform. The regular Kubernetes does not support this behavior.
- The maximum block duration (`InhibitDelayMaxSec`) is `3` days. After this period expires, the shutdown may proceed regardless of whether there are pods on the node with the label `pod.deckhouse.io/inhibit-node-shutdown`.
- This mechanism does not cancel the scheduling of disruptive updates in `node-manager`. It only delays the actual reboot of the node at the time of shutdown.

> ⚠ To decide whether to block a node shutdown, Deckhouse Platform additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

For information on enabling the delay mechanism, refer to the section "How to delay a node reboot while critical pods are running on it".

Automatic deploying, configuring and updating Kubernetes nodes

Automatic deployment (partially supported for *static/hybrid* nodes/clusters), configuration, and continuing software updates are supported for all cluster types (cloud or bare metal based).

Deploying Kubernetes nodes

Deckhouse automatically deploys cluster nodes by performing the following **idempotent** operations:

- Configuring the OS and optimizing it for containerd and Kubernetes:
  - Installing the needed packages from the distribution's repository.
  - Configuring kernel parameters, logging, log rotation, and other system parameters.
- Installing the appropriate versions of containerd and kubelet; adding the node to the Kubernetes cluster.
- Configuring kubernetes-api-proxy and updating the list of upstream resources for balancing node requests to the Kubernetes API.

Keeping nodes up-to-date

Two types of updates can be applied to keep cluster nodes up to date:

- **Regular**. These updates are always applied automatically and do not cause the node to stop or reboot.
- **Disruptive**. An example of such updates is a kernel or containerd version update, a major kubelet version change, etc. For this type of updates you can choose manual or automatic mode (disruptions parameter section). In automatic mode, the node is gracefully drained before the update.

Only one node of the group is updated at a time, and only if all the nodes in the group are available.

The `node-manager` module has a set of built-in metrics for monitoring the update process, alerting about issues with the update, or when a decision to proceed needs to be made by an administrator.

Working with nodes on supported cloud platforms

Each supported cloud provider can provision nodes in an automated manner. For this, you have to specify the required parameters for each node or a group of nodes.

Depending on the provider, these parameters can include:

- type of a node, the number of processor cores, the amount of RAM;
- disk size;
- security settings;
- connected networks;
- other parameters.

Creating, starting, and connecting virtual machines to the cluster are performed automatically.

Scaling nodes in the cloud

Two methods of scaling nodes in a group are supported:

- **Automatic scaling**

  If there is a resource shortage or pods in the `Pending` state are present, nodes will be added to the group. If there is no load on one or more nodes, they will be removed from the cluster. Autoscaling takes into account the group priority (the group with the highest priority will be scaled first).

  To enable automatic node scaling, specify *different* non-zero values for the minimum and maximum number of nodes in the group.

- **Fixed number of nodes**

  In this case, Deckhouse will maintain the specified number of nodes (e.g., by provisioning new nodes when nodes fail).

  To set a fixed number of nodes in a group and disable automatic scaling, specify the *same* values for minPerZone and maxPerZone.

Working with static nodes

When working with static nodes, some features of the `node-manager` module are limited:

- **No node provisioning.** Resources (bare metal servers, virtual machines, linked resources) are provisioned manually. Subsequent configuration of resources (connecting a node to the cluster, setting up monitoring, etc.) is performed either fully (similar to nodes in the cloud) or partly automatically.
- **No node autoscaling.** Maintaining the specified number of nodes in a group is supported when using Cluster API Provider Static (parameter staticInstances.count). That is, Deckhouse will attempt to maintain the specified number of nodes in the group, cleaning up unnecessary nodes and setting up new ones as needed (by picking them from StaticInstance resources that are in *Pending* state).

Configuring/clearing up a node, joining it to a cluster, and disjoining it can be done in the following ways:

- **Manually** using the pre-made scripts.

  To configure the server (VM) and join a node to the cluster, a special bootstrap script must be downloaded and run. This script is generated for each group of static nodes (each `NodeGroup` resource). It is located in the `d8-cloud-instance-manager/manual-bootstrap-for-<NAME-NODEGROUP>` secret. See an example of how to add a static node to a cluster.

  To disjoin a cluster node and clean up the server (virtual machine), you need to run the `/var/lib/bashible/cleanup_static_node.sh` script (it is present on each static node). An example of decommissioning a cluster node and cleaning up the server can be found in FAQ.

- **Automatically** using Cluster API Provider Static.

  Cluster API Provider Static (CAPS) connects to the server (VM) using StaticInstance and SSHCredentials resources, configures, and joins the node into the cluster.

  If necessary (for example, if the StaticInstance resource associated with the server is deleted or the number of group nodes is reduced), the Cluster API Provider Static connects to the cluster

node, clears it, and disconnects it from the cluster.

- **Manually with subsequent transfer of the Node under automatic control** of Cluster API Provider Static.

  > **i** This feature is available starting with Deckhouse 1.63.

  To transfer an existing cluster node under CAPS management, it is necessary to prepare the StaticInstance and SSHCredentials resources for this node, as with automatic management in the point above, however the StaticInstance resource must additionally be annotated as `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Cluster API Provider Static

Cluster API Provider Static (CAPS) is an implementation of a declarative management provider for static nodes (bare metal servers or virtual machines) for the Kubernetes Cluster API. Essentially, CAPS is an additional layer of abstraction to the existing Deckhouse functionality that provides automatic static node configuration and cleanup using scripts generated for each node group (see Working with Static Nodes).

CAPS features are as follows:

- configuring a bare metal server (or virtual machine) to join a Kubernetes cluster;
- joining a node to the Kubernetes cluster;
- disjoining the node from the Kubernetes cluster;
- cleaning the bare metal server (or virtual machine) after disjoining the node from the Kubernetes cluster.

CAPS uses the following CustomResource when operating:

- **StaticInstance.** Each `StaticInstance` resource details a specific host (server, VM) that is managed using CAPS.
- **SSHCredentials**. Contains the SSH credentials required to connect to the host (`SSHCredentials` data is specified in the credentialsRef parameter of the `StaticInstance` resource).
- **NodeGroup**. The staticInstances parameter section defines the required number of nodes in the group and the filter for the `StaticInstance` resources that can be used in the group.

CAPS is enabled automatically if the NodeGroup has the staticInstances parameter section filled in. If the `staticInstances` parameter section in the `NodeGroup` is empty, then setting up and cleaning up nodes for that group is done *manually* rather than using CAPS (see adding a static node to a cluster and cleaning up a node).

The workflow for dealing with static nodes when using Cluster API Provider Static (CAPS) (see the adding a node example) is as follows:

1. **Preparing resources.**

   Before bringing a bare metal server or virtual machine under CAPS control, some preliminary steps may be necessary:

   - Preparing the storage system, adding mount points, and so on.
   - Installing the OS-specific packages.
   - Configuring the network. For example, configuring the network between the server and the cluster nodes.
   - Configuring SSH access to the server, creating a user that is a member of sudoers group. A good practice is to create a separate user and unique keys for each server.

2. **Creating a SSHCredentials resource.**

   The `SSHCredentials` resource contains the parameters required by CAPS to connect to a server via SSH. A single `SSHCredentials` resource can be used to connect to multiple servers, but it is a good practice to create unique users and access keys to connect to each server. In this case, the `SSHCredentials` resource will also be unique for each server.

3. **Creating a StaticInstance resource.**

   For every server (VM) in the cluster, an individual `StaticInstance` resource is created. It contains the IP address for connecting and a link to the `SSHCredentials` resource with the data to be used for connecting.

   The following is a list of possible `StaticInstance` states and its associated servers (VMs) and cluster nodes:

  - `Pending`. The server is not configured and there is no associated node in the cluster.
  - `Bootstrapping`. The procedure for configuring the server (VM) and connecting the node to the cluster is in progress.
  - `Running`. The server is configured and the associated node is added to the cluster.
  - `Cleaning`. The procedure of cleaning up the server and disconnecting the node from the cluster is in progress.

> **i** You can transfer the existing manually-bootstrapped cluster node under CAPS management by annotating its StaticInstance with `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

4. **Creating a NodeGroup resource.**

   When using CAPS, you have to focus on the nodeType parameter (must be `Static`) of the `NodeGroup` resource as well as the staticInstances parameter section.

   The staticInstances.labelSelector parameter section defines a filter that CAPS applies to select the `StaticInstance` resources to be used for a group. The filter allows only certain `StaticInstance` to be used for specific node groups, and also allows a single `StaticInstance` to be used in different node groups. You can choose not to define a filter to use any available `StaticInstance` for a node group.

   The staticInstances.count parameter specifies the desired number of nodes in the group. When the parameter changes, CAPS starts adding or removing the desired number of nodes (this process runs in parallel).

Using the data in the staticInstances parameter section, CAPS attempts to maintain the specified number of nodes in the group (count parameter). If a node needs to be added to the group, CAPS selects the resource StaticInstance that matches the filter and is in the `Pending` state, configures the server (VM), and joins the node to the cluster. If a node needs to be removed from the group, CAPS selects the StaticInstance that is in the `Running` state, cleans up the server (VM) and disconnects the node from the cluster (the corresponding `StaticInstance` then goes to the `Pending` state and can be reused).

Custom node settings

The NodeGroupConfiguration resource allows you to automate actions on group nodes. It supports running bash scripts on nodes (you can use the bashbooster command set) as well as the Go Template templating engine, and is a great way to automate operations such as:

- Installing and configuring additional OS packages.

  Examples:

  - installing the plugin for kubectl.

- Updating the OS kernel to a specific version.

  Examples:

  - Debian kernel update;
  - CentOS kernel update.

- Modifying OS parameters.

  Examples:

  - customizing the sysctl parameter;
  - adding a root certificate.

- Collecting information on a node and carrying out other similar tasks.

The `NodeGroupConfiguration` resource allows you to assign priority to scripts being run or limit them to running on specific node groups and OS types.

The script code is stored in the content of the resource. When a script is created on a node, the contents of the `content` parameter are fed into the Go Template templating engine. The latter embeds an extra layer of logic when generating a script. When parsed by the templating engine, a context with a set of dynamic variables becomes available.

The following variables are supported by the templating engine:

- `.cloudProvider` (for node groups of nodeType `CloudEphemeral` or `CloudPermanent`) — cloud provider dataset.

Example of data...

```
cloudProvider:
  instanceClassKind: OpenStackInstanceClass
  machineClassKind: OpenStackMachineClass
  openstack:
    connection:
      authURL: https://cloud.provider.com/v3/
      domainName: Default
      password: p@ssw0rd
      region: region2
      tenantName: mytenantname
      username: mytenantusername
    externalNetworkNames:
    - public
    instances:
      imageName: ubuntu-24-04-cloud-amd64
      mainNetwork: kube
      securityGroups:
      - kube
      sshKeyPairName: kube
    internalNetworkNames:
    - kube
    podNetworkMode: DirectRoutingWithPortSecurityEnabled
  region: region2
  type: openstack
  zones:
  - nova
```

- `.cri` — the CRI in use (starting with Deckhouse 1.49, only `Containerd` is supported).
- `.kubernetesVersion` — the Kubernetes version in use.
- `.nodeUsers` — the dataset with information about node users added via the NodeUser.

  Example of data...

  ```
  nodeUsers:
  - name: user1
    spec:
      isSudoer: true
      nodeGroups:
      - '*'
      passwordHash: PASSWORD_HASH
      sshPublicKey: SSH_PUBLIC_KEY
      uid: 1050
  ```

- `.nodeGroup` — node group dataset.

  Example of data...

```
nodeGroup:
  cloudProviderType: openstack
  cri:
    type: Containerd
  disruptions:
    approvalMode: Automatic
  kubelet:
    containerLogMaxFiles: 4
    containerLogMaxSize: 50Mi
    resourceReservation:
      mode: "Off"
  kubernetesVersion: "1.31"
  manualRolloutID: ""
  name: master
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
    taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  nodeType: CloudPermanent
  updateEpoch: "1699879470"
```

An example of using variables in a template:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

An example of using bashbooster commands:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

The progress of script execution can be seen on the node in the bashible service log using the command:

```
journalctl -u bashible.service
```

The scripts themselves are located on the node in the directory `/var/lib/bashible/bundle_steps/`.

The service decides to re-run the scripts by comparing the single checksum of all files located at `/var/lib/bashible/configuration_checksum` with the checksum located in the `kubernetes` cluster in the `configuration-checksums` secret namespace `d8-cloud-instance-manager`.

You can see the checksum using the following command:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o yaml
```

The service compares checksums every minute.

The checksum in the cluster changes every 4 hours, thereby re-running the scripts on all nodes. Forcing the execution of bashible on a node can be done by deleting the file with the script checksum using the following command:

```
rm /var/lib/bashible/configuration_checksum
```

Features of writing scripts

When writing your own scripts, it is important to consider the following features of their use in Deckhouse:

1. Scripts in deckhouse are executed once every 4 hours or more often based on external triggers. Therefore, it is important to write scripts in such a way that they check the need for their changes in the system before performing actions, thereby not making changes every time they are launched.
2. There are built-in scripts that perform various actions, including installing and configuring services. This is important to consider when choosing the priority of custom scripts. For example, if a script is planned to restart a service, then this script should be called after the service installation script, since otherwise it will not be able to execute when deploying a new node.

Useful features of some scripts:

- `032_configure_containerd.sh`: Merges the containerd configuration files and **restarts** the service. The location of the configuration files depends on the CRI used on the cluster nodes:
  - `/etc/containerd/conf.d/*.toml`: For containerd v1.
  - `/etc/containerd/conf2.d/*.toml`: For containerd v2.

  Note that directories containing configuration files are not created automatically. Files must be added to these directories in scripts with a priority of less than `32`.

Chaos Monkey

The instrument (you can enable it for each `NodeGroup` individually) for unexpected and random termination of nodes in a systemic manner. Chaos Monkey tests the resilience of cluster elements, applications, and infrastructure components.

Monitoring

Deckhouse Platform exports the group availability Prometheus metrics for each `NodeGroup`.

What information does Prometheus collect and in what form?

Metrics have the `d8_node_group_` prefix.

All the collected metrics have a label that allow you to identify the node group: `node_group_name`.

The following metrics are collected for each node group:

- `d8_node_group_ready` — the number of Kubernetes nodes in the group that are `Ready`.
- `d8_node_group_nodes` — the number of Kubernetes nodes (in any state) in the group.
- `d8_node_group_instances` — the number of instances (in any state) in the group.
- `d8_node_group_desired` — the number of desired `Machines` in the group.
- `d8_node_group_min` — the minimum number of instances in the group.
- `d8_node_group_max` — the maximum number of instances in the group.
- `d8_node_group_up_to_date` — the number of up-to-date nodes in the group.
- `d8_node_group_standby` — the number of over provisioned instances (see the `standby` parameter) in the group.
- `d8_node_group_has_errors` — equals to 1 if there are errors in the group.

Third-party components

List of third-party software used in the `node-manager` module:

- Gardener Machine Controller Manager v0.36.0

  License: Apache License 2.0

  An operator that manages VMs as a kubernetes custom resource.

- Cluster API 1.12.11

  License: Apache License 2.0

  Kubernetes Cluster API for declarative cluster lifecycle management.

- Cluster Autoscaler v1.33.1, v1.34.4, v1.35.2, v1.36.1, v1.36.1

  License: Apache License 2.0

  Kubernetes autoscaler that automatically adjusts the size of the cluster.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing nodes: configuration

> The module lifecycle stage: General Availability

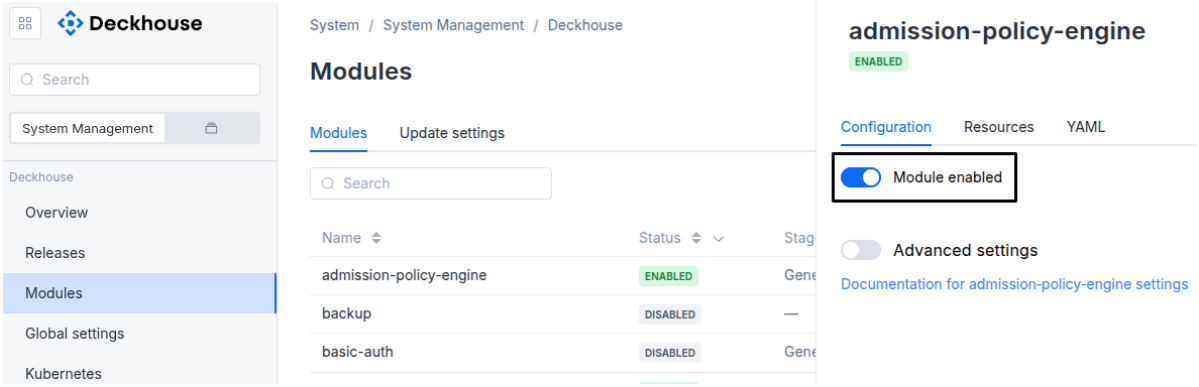
The module has 61 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable node-manager
```

- **Using ModuleConfig `node-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `node-manager` (create it if necessary);

  Example of a manifest to enable module `node-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  enabled: true
```

How to configure the module...

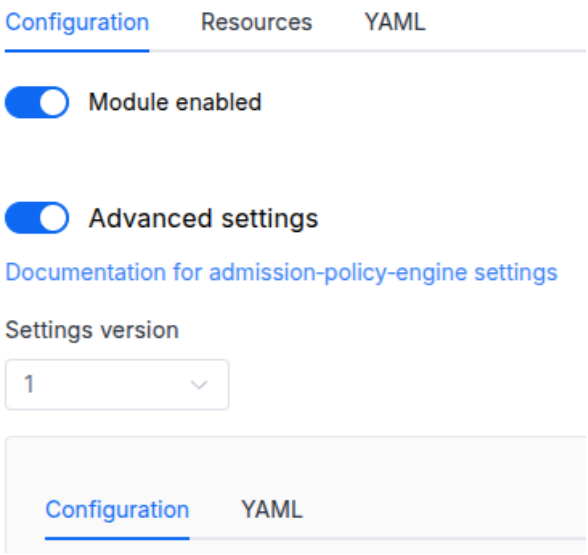
You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-`

`manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `node-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `node-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `node-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc node-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `node-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `node-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: node-manager
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

Delete `allowedKubernetesVersions` and `allowedBundles`.

- Updates from version **2** to **3**:

  Delete `earlyOomEnabled`.

Parameters

Schema version: 3

- **settings**
  object
  - **settings.capiEmergencyBrake**
    boolean

    Set to ‘true’ to disable CAPI in case of potentially cluster-damaging changes.

    > **Warning.** Use this flag only if you fully understand the consequences.

    Default: `false`

    Example:

    ```
    capiEmergencyBrake: false
    ```

  - **settings.fencing**
    object

    Available in editions: Ultimate/EE

    Fencing settings.

    - **settings.fencing.systemNetworkFallback**
      object

      Settings of the second network of the fencing agents.

      - **settings.fencing.systemNetworkFallback.systemNetworkName**
        string

        The name of the SystemNetwork object for the second network of the fencing agents. Currently the agents do not exchange messages through this network.

        An empty value or no value disables the second network.

        Deckhouse checks that the SystemNetwork object exists when you set or change the value. If you delete the object later, the `D8FencingSystemNetworkNotFound` alert rises.

        Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)?$`

        Maximum length: `253`

        Example:

        ```
        systemNetworkName: sysnet-a
        ```

  - **settings.instancePrefix**
    string

    The prefix to use when creating instances via the corresponding cloud provider module.

    The default value can be calculated based on the `ClusterConfiguration` struct if the cluster is installed via the Deckhouse installer.

    Example:

    ```
    instancePrefix: kube
    ```

  - **settings.mcmEmergencyBrake**
    boolean

Available in editions: Open/CE, BE, SE, SE+, Ultimate/EE

Set to ‘true’ to disable machine-controller-manager in case of potentially cluster-damaging changes.

> **Warning.** Use this flag only if you fully understand the consequences.

Default: `false`

Example:

```
mcmEmergencyBrake: false
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing nodes: custom resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

FencingFailedNodeState

Short names: ffns

Scope: Cluster
Version: v1alpha1

Fencing incident of one node.

The name of the object is the name of the failed node. The object has exactly one owner reference to this node. When the Node object is deleted, the Kubernetes garbage collector deletes this object too. The controller does nothing for an object whose owner reference does not have the UID of the node with this name, for example after the node is created again with the same name. A healthy node has no `FencingFailedNodeState`: the object exists only while the node has an active incident.

A fencing agent creates the object. The agent of any node can create it, also the agent on a node of a group in the `Disabled` mode. The creator writes `spec`. Nobody can change `spec` after that. Each part of `status` has one owner. The fencing controller reads the object and decides if it must evacuate the node.

The object is deleted in these cases:

- The failed node comes back with the same boot ID before the phase `Proven`. Then the fencing agents delete the object.
- The Node is `Ready` and has a new boot ID. Then the fencing controller removes the `node.kubernetes.io/out-of-service` taint from the Node and deletes the object.
- The Node object is deleted. Then the Kubernetes garbage collector deletes the object.

Do not create the object manually. Do not edit `spec` or `status`.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Required value

  Immutable description of the incident. The creator of the object writes it at creation. You cannot change it after that.

  - **spec.armed**
    boolean

    Required value

    `true` if the watchdog of the failed node was armed at the time of the failure. A fencing agent takes the value from the memberlist metadata of the node.

  - **spec.bootID**
    string

Required value

Boot ID of the failed node at the time of the failure. A fencing agent takes it from the memberlist metadata of the node. For the `NodeLeaseStale` source, the controller takes it from `status.nodeInfo.bootID` of the Node.

- **spec.detectedAt**
  string

  Required value

  Time when the creator detected the failure, with microsecond precision.

  The value is for reference only. The fencing controller counts its deadlines from its own observations.

  Pattern: `^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{6}(Z|[+-][0-9]{2}:[0-9]{2})$`

- **spec.detectedBy**
  string

  Required value

  Name of the node whose agent created the object. If the fencing controller created the object, the value is `fencing-controller`.

- **spec.profileRef**
  object

  SLA profile of the node group of the failed node at the time when the object was created. The group is the value of the `node.deckhouse.io/group` label of the Node. The mode of the group selects the profile. If the group is in the `Disabled` mode, the value is the profile of the memberlist network of the cluster.

  The field is set only if `armed` is `true`. The fencing controller takes the timings of the failed node only from this profile.

  - **spec.profileRef.name**
    string

    Required value

    Name of the built-in SLA profile. The value can be only the name of a built-in profile.

    The name of the related `FencingSLAProfile` object is this value in lower case, because the name of a Kubernetes object cannot have capital letters. To find the profile, the controller changes this value to lower case.

    Allowed values: `Critical`, `Normal`, `Slow`

- **spec.source**
  string

  Required value

  Source of the incident. It tells who created the object:

  - `MemberlistDead`: a fencing agent that saw the node as dead in the memberlist network of the cluster.
  - `DeclareSuicide`: a fencing agent that received the `DeclareSuicide` message of the node. This agent is the witness.
  - `NodeLeaseStale`: the fencing controller, because the Lease of the node is stale. The current version of the controller does not create objects with this source.

  Allowed values: `MemberlistDead`, `DeclareSuicide`, `NodeLeaseStale`

- **status**
  object

  Observed state of the incident. Each part of the status has one owner:

  - `attestation`: the fencing agent of the witness node.
  - `contact`: the fencing agent that received a report of a contact with the failed node. The current version of the agent does not write this section.
  - `fallback`: the fencing agent of the failed node.
  - `phase`, `correlation` and `conditions`: the fencing controller. The current version of the controller does not write `correlation`.

The status is a subresource. The creator of the object cannot set it in the create request.

- **status.attestation**
  object

  Attestation of the failure. The witness is a node whose agent received the `DeclareSuicide` message of the failed node. The agent of the witness writes the name of its own node.

  - **status.attestation.at**
    string

    Required value

    Time of the attestation, with microsecond precision.

    Pattern: `^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{6}(Z|[+-][0-9]{2}:[0-9]{2})$`

  - **status.attestation.witness**
    string

    Required value

    Name of the witness node.

- **status.conditions**
  array of objects

  Conditions of the incident. A condition does not replace the phase, so the controller can report the same problem in any phase.

  The controller sets `ConfigurationError` with the status `True` and the reason `ProfileUnavailable` while it cannot read the timings that it needs from the SLA profile in `spec.profileRef.name`. The message of the condition tells the cause. In this state, the controller does not evacuate the node. The status `False` with the reason `ProfileResolved` tells that the controller found the profile. An incident with `armed: false` needs no profile and also has this status.

  The `Queued` condition with the status `True` tells why the evacuation waits in the `Queued` phase. The reasons are `ConcurrencyLimit` and `MassFailure`. The current version of the controller does not use this condition.

  The controller sets `Progressing` with the status `False` when the evacuation in the `Evicting` phase is not complete after 60 seconds. The controller continues to examine the node. The `reason` tells what stops the evacuation now, and the `message` gives the details:

  - `VolumesAttached`: the node still has `VolumeAttachment` objects, so the volumes of its pods are not free for other nodes. The message gives their names.
  - `PodsRemaining`: the controller could not delete pods of the node, or another error of the API stops the evacuation. The message gives the pods and the errors.
  - `TooManyUnhealthyMachines`: the controller does not delete the Node of a cloud CAPI machine, because the MachineHealthCheck does not replace the machine now: more than 50 % of the machines are unhealthy, or the cluster has no MachineHealthCheck that reports its machines. The message gives the numbers.

  When the evacuation is complete, the phase is `Evicted`, and the controller removes the `Progressing` condition.

  - **status.conditions.lastTransitionTime**
    string

    Required value

    Time the condition last changed its status.

  - **status.conditions.message**
    string

    Human-readable description of the last transition.

  - **status.conditions.observedGeneration**
    integer

    Generation of the resource the condition was set from.

    Allowed values: `0 <= X`

  - **status.conditions.reason**

string

Required value

Machine-readable reason for the last transition.

- **status.conditions.status**
  string

  Required value

  Condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type.

- **status.contact**
  object

  Last contact with the failed node. A fencing agent that received a report of a contact with the node writes this section.

  - **status.contact.at**
    string

    Required value

    Time of the last contact with the failed node, with microsecond precision.

    Pattern: `^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{6}(Z|[+-][0-9]{2}:[0-9]{2})$`

  - **status.contact.reportedBy**
    string

    Required value

    Name of the node that reported the contact.

- **status.correlation**
  object

  Correlation of the incident with the incidents of other nodes. The current version of the controller does not write this section.

  - **status.correlation.episode**
    string

    Identifier of the episode of correlated failures that includes this incident.

- **status.fallback**
  object

  Liveness evidence from the failed node. Only the fencing agent of the failed node writes this section, and only through the Kubernetes API. Thus the section exists only if the node reached the API.

  The fencing controller keeps the `FallbackAlive` phase while the last heartbeat is not older than `2 * heartbeat.interval + heartbeat.apiTimeout` of the profile.

  - **status.fallback.lastHeartbeatAt**
    string

    Required value

    Time of the last heartbeat of the node, with microsecond precision.

    When fresh heartbeats stop, this is the signal that the node cannot reach the API.

    Pattern: `^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{6}(Z|[+-][0-9]{2}:[0-9]{2})$`

- **status.phase**

string

Phase of the incident:

- `Suspected`: a failure of the node is suspected.
- `FallbackAlive`: the node writes heartbeats through the Kubernetes API. For example, an ACL or a firewall blocks the network between the nodes, but the node can reach the API.
- `Proven`: the failure is proven. The controller proves a failure in one of two ways:
  - attestation: a witness confirmed the stop of the node, and the time `watchdog.timeout` + `watchdog.margin` of the profile passed after the controller saw the attestation. The node did not send a heartbeat after the attestation;
  - silence of the node: the time `controlPlaneLostAfter` + 0.2s + `watchdog.timeout` + `watchdog.margin` + 0.9s of the profile passed after the last sign of life of the node. The last sign of life is the later of two moments: when the controller saw the object first, and when it saw the last heartbeat plus `heartbeat.interval`.

  While the `Fencing` condition of the Node is not `True`, the controller does not write `Proven`, and the object stays in the phase `Suspected`.

- `Unproven`: the failure is a part of a correlated failure and has no attestation. The current version of the controller does not set this phase.
- `Queued`: the evacuation waits. The `Queued` condition tells the reason. The current version of the controller does not use this phase.
- `Evicting`: the node has the `node.kubernetes.io/out-of-service` taint, and the evacuation is in progress. If the evacuation is not complete after 60 seconds, the `Progressing` condition tells the reason.
- `Evicted`: the evacuation is complete: the controller deleted the pods of the node, and the node has no `VolumeAttachment` objects.

The controller writes `Evicted` before it deletes the Node or the Machine of a node that the cloud replaces. The deletion of the Node also deletes this object. If the write of `Evicted` fails, the controller does not delete the Node or the Machine. If the deletion fails, the controller sends it again in the `Evicted` phase.

An object with `armed: false` never gets the phase `Proven`. It stays in the phase `Suspected` or `FallbackAlive`, and the node never gets the taint.

An empty phase means that the controller did not process the object yet. Read it as `Suspected`.

An error is not a phase. The controller reports an error in `conditions` and keeps the phase.

Allowed values: `Suspected`, `FallbackAlive`, `Proven`, `Unproven`, `Queued`, `Evicting`, `Evicted`

FencingSLAProfile

Short names: fsp

Scope: Cluster
Version: v1alpha1

SLA profile of fencing. A profile holds only timings. Correlation, the brake and the eviction limit are settings of the module, not of a profile.

The module ships three profiles: `critical`, `normal` and `slow`. Community Edition (CE) has no `critical` profile, because the `Critical` mode is available only in the commercial editions. Only Deckhouse can change the profiles. NodeGroup `spec.fencing.mode` selects the profile of a group. The mode name in lower case is the name of the object, because the name of a Kubernetes object cannot have capital letters.

All agents of the cluster run one memberlist network. They use the `memberlist` and `rejoin` sections of the profile of the fastest group that is not `Disabled`. An agent of a group that is not `Disabled` takes the other sections from the profile of its own group.

A profile is a cluster-scoped object. It has no state, so the resource has no `status`.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object

- **spec**
  object

  Required value

  Timings of the profile. Every value is a duration or a count. The CEL rules of this object check the inequalities of one profile; the agent checks the others.

  The profile has no probe timings of kubernetes-api-proxy. The proxy uses the same fixed values on all nodes:

  - probe interval: 1.2s at most (1s with a jitter of 20 %);
  - probe timeout: 0.5s;
  - no retry of a failed probe.

  The largest age of a probe is 1.7s (1.2s + 0.5s).

  The profile has no timeouts of the poll of the control plane state and no read period of kubernetes-api-proxy. These timings are the same for all profiles: they are internal values of the module.

  - **spec.controlPlaneLostAfter**
    string

    Required value

    Maximum age of the last successful control plane probe. The node sees the control plane while the last `Ready` answer of any apiserver upstream is not older than this value.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.eviction**
    object

    Required value

    Parameters of the creation of `FencingFailedNodeState` objects.

    The profile has no delay before the taint of a silent node, a node without signs of life. The fencing controller calculates it from the other values of the profile: `controlPlaneLostAfter` + 0.2s + `watchdog.timeout` + `watchdog.margin` + 0.9s. The controller counts the delay from the last sign of life of the node.

    - **spec.eviction.writerWaveInterval**
      string

      Required value

      Interval between the waves of the agents that create `FencingFailedNodeState` objects.

      Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

      Maximum length: `32`

  - **spec.heartbeat**
    object

    Required value

    Parameters of the heartbeat that a failed node writes through the Kubernetes API.

    - **spec.heartbeat.apiTimeout**
      string

      Required value

      Timeout of one request of the agent to the Kubernetes API. The memberlist TCP timeout uses it too.

      Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

      Maximum length: `32`

    - **spec.heartbeat.interval**
      string

      Required value

Interval between the heartbeat writes of a failed node into its own `FencingFailedNodeState`.

Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

Maximum length: `32`

- **spec.memberlist**
  object

  Required value

  Parameters of memberlist and Lifeguard. All agents of the cluster use this section of the fastest profile among the node groups that are not `Disabled`.

  - **spec.memberlist.awarenessMaxMultiplier**
    integer

    Required value

    Maximum slowdown of the probe interval of a locally degraded agent (Lifeguard awareness).

    Allowed values: `1 <= X`

  - **spec.memberlist.deadNodeReclaimTime**
    string

    Required value

    Time after which a new node can take the name or the address of a dead node.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.memberlist.gossipInterval**
    string

    Required value

    Interval between gossip messages.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.memberlist.gossipNodes**
    integer

    Required value

    Number of peers that receive gossip in each gossip interval.

    Allowed values: `1 <= X`

  - **spec.memberlist.gossipToTheDeadTime**
    string

    Required value

    Time during which a dead node still receives gossip. The node can refute the dead state after it recovers.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.memberlist.indirectChecks**
    integer

    Required value

    Number of indirect checks when the direct probe of a peer fails.

    Allowed values: `1 <= X`

  - **spec.memberlist.probeInterval**
    string

Required value

Interval between probes of one peer. A shorter interval gives faster detection and more probe traffic. The value must be at least twice `probeTimeout`.

Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

Maximum length: `32`

- **spec.memberlist.probeTimeout**
  string

  Required value

  Time to wait for the answer to a probe. The value must be at most half of `probeInterval`.

  Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

  Maximum length: `32`

- **spec.memberlist.retransmitMult**
  integer

  Required value

  Retransmit multiplier of gossip broadcasts: `retransmitMult * ceil(log10(nodeCount + 1))`, where `nodeCount` is the number of nodes.

  Allowed values: `1 <= X`

- **spec.memberlist.suspicionMaxTimeoutMult**
  integer

  Required value

  Upper bound of the suspicion timer when no confirmations arrive.

  Allowed values: `1 <= X`

- **spec.memberlist.suspicionMult**
  integer

  Required value

  Multiplier of the suspicion window before memberlist marks a peer as dead. The value must be at least 3.

  Allowed values: `1 <= X`

- **spec.poll**
  object

  Required value

  Parameters of the poll of the control plane state between agents.

  - **spec.poll.witnessFresherBy**
    string

    Required value

    Minimum gap between the own last successful probe and the probe of the answering node. An answer with a smaller gap is not a witness.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

- **spec.rejoin**
  object

  Required value

  Parameters of the join attempts of the agent. All agents of the cluster use this section of the fastest profile among the node groups that are not `Disabled`.

  - **spec.rejoin.interval**
    string

Required value

Interval between the join attempts of the agent. The agent repeats the attempt at this fixed interval while some nodes are missing from the memberlist network.

Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

Maximum length: `32`

- **spec.watchdog**
  object

  Required value

  Parameters of the watchdog of the node.

  - **spec.watchdog.feedInterval**
    string

    Required value

    Interval between two feeds of the watchdog. The value must be at most a quarter of `timeout`.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.watchdog.margin**
    string

    Required value

    Reserve that is added to `timeout`. The value must be at least 300ms and at least 5 % of `timeout`.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

  - **spec.watchdog.timeout**
    string

    Required value

    Timeout of the watchdog. The node panics when the agent does not feed the watchdog for this time. The value is a whole number of seconds, because the watchdog device takes whole seconds.

    Pattern: `^([0-9]+(\.[0-9]+)?(ms|s|m|h))+$`

    Maximum length: `32`

Instance

Scope: Cluster

- v1alpha2
- v1alpha1

Instance describes the full lifecycle of a server in Deckhouse Platform, aggregating status from the Machine object (infrastructure) and Bashible service on the node (configuration).

- **spec**
  object

  Specification of the Instance, including references to related resources.

  - **spec.classReference**
    object

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` or `infra-provider-*` module.

    - **spec.classReference.kind**
      string

      The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the

documentation of the corresponding cloud or infrastructure provider module.

- **spec.classReference.name**
  string

  The name of the required `InstanceClass` object (e.g., `finland-medium`).

- **spec.machineRef**
  object

  Reference to instance in the cloud.

  - **spec.machineRef.apiVersion**
    string

    APIVersion defines the versioned schema of this representation of an reference object.

  - **spec.machineRef.kind**
    string

    Kind is a string value representing the REST resource referenced object represents.

  - **spec.machineRef.name**
    string

    Name of referenced object.

  - **spec.machineRef.namespace**
    string

    Namespace of referenced object.

- **spec.nodeGroupRef**
  object

  Reference to the NodeGroup that ordered this instance. It is known before the node joins the cluster.

  - **spec.nodeGroupRef.name**
    string

    NodeGroup name.

- **spec.nodeRef**
  object

  Reference to the kubernetes node object.

  - **spec.nodeRef.name**
    string

    Node name.

- **status**
  object

  Most recently observed status of the Instance.

  - **status.bashibleStatus**
    string

    Aggregated status of the bashible configuration process.

    Derived from `BashibleReady` condition.

    Allowed values: `Error`, `Unknown`, `Ready`, `WaitingApproval`

  - **status.bootstrapStatus**
    object

    Information about instance bootstrapping process.

    - **status.bootstrapStatus.description**
      string

      Description about getting logs.

    - **status.bootstrapStatus.logsEndpoint**
      string

http endpoint for getting bootstrap logs.

- **status.conditions**
  array of objects

  Detailed conditions of the instance (`MachineReady`, `BashibleReady`, etc.).

  On a node that has a NodeConfig, `NodeAgentReady` is `True` while the node agent reports and `Unknown` with reason `NoReport` after 10 minutes without a report.

  - **status.conditions.lastHeartbeatTime**
    string

    Last time the condition heartbeat was updated by its source.

  - **status.conditions.lastTransitionTime**
    string

    Last time the condition transitioned from one status to another.

  - **status.conditions.message**
    string

    A human-readable message indicating details about the transition.

  - **status.conditions.observedGeneration**
    integer

    The generation observed by the controller.

  - **status.conditions.reason**
    string

    Required value

    The reason for the condition’s last transition.

  - **status.conditions.severity**
    string

    Severity of the condition (`Error`, `Warning`, `Info`).

  - **status.conditions.status**
    string

    Required value

    Status of the condition (`True`, `False`, `Unknown`).

  - **status.conditions.type**
    string

    Required value

    Type of the condition (e.g., `MachineReady`, `BashibleReady`).

- **status.machineStatus**
  string

  Aggregated status of the infrastructure machine.

  Derived from `MachineReady` condition.

  Allowed values: `Progressing`, `Error`, `Ready`, `Blocked`, `Rebooting`

- **status.message**
  string

  A human-readable message indicating details about the current status or error.

  Displayed in CLI only when using `-o wide` for `d8 k`.

- **status.phase**
  string

  The high-level lifecycle phase of the Instance.

  Mapping: Pending -> Provisioning -> Provisioned -> Running -> Terminating.

Allowed values: `Pending`, `Provisioning`, `Provisioned`, `Running`, `Terminating`, `Unknown`

Describes an implementation-independent ephemeral machine resource.

- **status**
  object
  - **status.bootstrapStatus**
    object

    Information about instance bootstrapping process.

    - **status.bootstrapStatus.description**
      string

      Description about getting logs.

    - **status.bootstrapStatus.logsEndpoint**
      string

      http endpoint for getting bootstrap logs.

  - **status.classReference**
    object

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **status.classReference.kind**
      string

      The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the documentation of the corresponding `cloud-provider-` module.

      Allowed values: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`, `BareMetalInstanceClass`

    - **status.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **status.currentStatus**
    object

    Current status of the instance object.

    - **status.currentStatus.lastUpdateTime**
      string

      Last update time of current status.

    - **status.currentStatus.phase**
      string

      InstancePhase is a label for the condition of a instance at the current time.

  - **status.lastOperation**
    object

    Last operation refers to the status of the last operation performed.

    - **status.lastOperation.description**
      string

      Description of the current operation.

    - **status.lastOperation.lastUpdateTime**
      string

      Last update time of current operation.

    - **status.lastOperation.state**
      string

      State of operation.

        - **status.lastOperation.type**
          string

          Type of operation.

    - **status.machineRef**
      object

      Reference to instance in the cloud.

        - **status.machineRef.apiVersion**
          string

          APIVersion defines the versioned schema of this representation of an reference object.

        - **status.machineRef.kind**
          string

          Kind is a string value representing the REST resource referenced object represents.

        - **status.machineRef.name**
          string

          Name of referenced object.

        - **status.machineRef.namespace**
          string

          Namespace of referenced object.

    - **status.nodeRef**
      object

      Reference to the kubernetes node object.

        - **status.nodeRef.name**
          string

          Node name.

InstanceTypesCatalog

Scope: Cluster
Version: v1alpha1

Contains information about discovered data from cloud.

- **instanceTypes**
  array of objects

  Required value

  List of available instance types.

    - **instanceTypes.cpu**
      integer or string

      Required value

      The number of CPUs for the instance type.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    - **instanceTypes.memory**
      integer or string

      Required value

      The amount of RAM a instance type has.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    - **instanceTypes.name**
      string

      Instance type name.

  - **instanceTypes.rootDisk**
    integer or string

    Root disk size.

    Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

NodeGroup

Short names: ng

Scope: Cluster

- v1
- v1alpha2
- v1alpha1

Describes the runtime parameters of the node group.

Example:

```
# NodeGroup for cloud nodes in AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
---
# NodeGroup for static nodes on bare metal servers (or VMs).
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`

- **spec**
  object

  Required value

  - **spec.chaos**
    object

    Chaos monkey settings.

Example:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

- **spec.chaos.mode**
  string

  The chaos monkey mode:

  - `DrainAndDelete` — drains and deletes a node when triggered;
  - `Disabled` — leaves this NodeGroup intact.

  Default: `Disabled`

  Allowed values: `Disabled`, `DrainAndDelete`

- **spec.chaos.period**
  string

  The time interval to use for the chaos monkey.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  Default: `6h`

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.cloudInstances**
  object

  Parameters of the node group cloud virtual machines.

  The `cloudInstances` parameter is used for `CloudEphemeral` nodes. When the cloud provider is configured through ModuleConfig, it is also used for `CloudPermanent` nodes, including master nodes. In this case, `classReference` specifies the instance class with the virtual machine parameters.

  - **spec.cloudInstances.classReference**
    object

    Required value

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **spec.cloudInstances.classReference.kind**
      string

      The kind of the `InstanceClass` object (e.g., `OpenStackInstanceClass`). Supported kinds are defined by the installed cloud provider modules and specified in their documentation.

    - **spec.cloudInstances.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **spec.cloudInstances.maxPerZone**
    integer

    Required value

    For `CloudEphemeral` nodes, sets the maximum number of nodes in each zone and the upper bound for autoscaling.

    For `CloudPermanent` nodes, the field is required when `cloudInstances` is set. Specify a value greater than or equal to `minPerZone`.

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxSurgePerZone**
    integer

    The maximum number of instances the platform creates in each zone above the desired

number while it replaces the instances of the group.

Replacement happens when parameters that require recreating a virtual machine change, for example in the InstanceClass. The parameter does not limit the number of instances that the cluster autoscaler orders when it scales the group up. For details on what NodeGroup parameters affect, see NodeGroup parameters and their result.

The value `0` is treated as if the parameter is not set, so the default value applies.

Default: `1`

Allowed values: `0 <= X`

- **spec.cloudInstances.maxUnavailablePerZone**
  integer

  The maximum number of unavailable instances (during rollout) in the group in each zone.

  Default: `0`

  Allowed values: `0 <= X`

- **spec.cloudInstances.minPerZone**
  integer

  Required value

  For `CloudEphemeral` nodes, sets the minimum number of nodes in each zone and the lower bound for autoscaling.

  For `CloudPermanent` nodes with a ModuleConfig-based provider configuration, sets the desired number of nodes in the group.

  Allowed values: `0 <= X`

- **spec.cloudInstances.priority**
  integer

  Priority of the node group.

  When scaling a cluster, the autoscaler will first select node groups with a higher priority set. If several node groups have the same priority, the autoscaler randomly selects a group of them. You can control the rate at which nodes are switched to lower-priority groups on demand (for more details, see the FAQ).

  Using priorities can be convenient to prefer ordering cheaper nodes (for example, spot instances) over more expensive ones.

- **spec.cloudInstances.quickShutdown**
  Deprecated
  boolean

  Lowers CloudEphemeral machine drain timeout to 5 minutes.

- **spec.cloudInstances.standby**
  integer or string

  The summary number of overprovisioned nodes for this `NodeGroup` in all zones.

  An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

  The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

  Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
  object

  Amount of reserved resources.

  Used to determine whether to order overprovisioned nodes.

- **spec.cloudInstances.standbyHolder.notHeldResources**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Use the `overprovisioningRate` parameter.

  Describes the resources that will not be held (consumed) by the standby holder.

  - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
    integer or string

    Describes the amount of CPU that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

    Pattern: `^[0-9]+m?$`

  - **spec.cloudInstances.standbyHolder.notHeldResources.memory**
    integer or string

    Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.standbyHolder.overprovisioningRate**
  integer

  Percentage of reserved resources calculated from the capacity of a node of a `NodeGroup`.

  Default: `50`

  Allowed values: `1 <= X <= 80`

- **spec.cloudInstances.zones**
  array of strings

  List of availability zones to create instances in.

  The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

  Example:

  ```
  zones:
  - Helsinki
  - Espoo
  - Tampere
  ```

- **spec.cri**
  object

  Container runtime parameters.

  - **spec.cri.containerd**
    object

    Containerd runtime parameters.

    If used, `cri.type` must be set to `Containerd`.

    - **spec.cri.containerd.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

      Default: `3`

  - **spec.cri.containerdV2**
    object

ContainerdV2 runtime parameters.

If used, `cri.type` must be set to `ContainerdV2`.

When using `ContainerdV2`, `CgroupsV2` will be used (providing improved security and resource management).

To use `ContainerdV2` as a container runtime, the nodes in the group must meet the following requirements:

- Support for `CgroupsV2`.
- Linux kernel version `5.8` or newer.
- Systemd version `244` or newer.
- Support for `erofs` kernel module.
- **spec.cri.containerdV2.maxConcurrentDownloads**
  integer

  Set the max concurrent downloads for each pull.

  Default: `3`

- **spec.cri.docker**
  Deprecated
  object

  Docker settings for nodes.

  - **spec.cri.docker.manage**
    boolean

    Enable Docker maintenance from bashible.

    Default: `true`

  - **spec.cri.docker.maxConcurrentDownloads**
    integer

    Set the max concurrent downloads for each pull.

    Default: `3`

- **spec.cri.notManaged**
  object

  Settings for not managed CRI for nodes.

  - **spec.cri.notManaged.criSocketPath**
    string

    Path to CRI socket.

- **spec.cri.type**
  string

  Container runtime type.

  If not specified, the `defaultCRI` parameter value of ClusterConfiguration is used.

  > **Note.** Docker support is deprecated.

  > **Note.** The CSE edition supports containerd v2 only.

  Allowed values: `Docker`, `Containerd`, `ContainerdV2`, `NotManaged`

- **spec.disruptions**
  object

  Disruptions settings for nodes.

  Example:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
    windows:
    - from: '06:00'
      to: '08:00'
      days:
      - Tue
      - Sun
```

- **spec.disruptions.approvalMode**
  string

  The approval mode for disruptive updates:

  - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. Nodes are not drained automatically in `Manual` mode. If necessary, manually drain nodes before setting the `update.node.deckhouse.io/disruption-approved=` annotation. **Caution.** The `master` node group update mode must be `Manual` to avoid issues with draining.
  - `Automatic` — automatically approve disruption-involving updates. Nodes could be drained automatically before reboot, as configured by the drainBeforeApproval option.
  - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

  If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

  Default: `Automatic`

  Allowed values: `Manual`, `Automatic`, `RollingUpdate`

- **spec.disruptions.automatic**
  object

  Additional parameters for the `Automatic` mode.

  - **spec.disruptions.automatic.drainBeforeApproval**
    boolean

    Drain Pods from the nodes before approving disruption.

    Caution. This setting ignores (nodes will be approved without draining Pods):

    - for the nodeGroup `master` with a single node;
    - for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

    Default: `true`

  - **spec.disruptions.automatic.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.automatic.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

```
days: Wed
```

- **Element of the array**
  string

  Day of the week.

  Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

- **spec.disruptions.automatic.windows.from**
  string

  Required value

  Start time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

```
from: '13:00'
```

- **spec.disruptions.automatic.windows.to**
  string

  Required value

  End time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

```
to: '18:30'
```

- **spec.disruptions.rollingUpdate**
  object

  Additional parameters for the `RollingUpdate` mode.

  - **spec.disruptions.rollingUpdate.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.rollingUpdate.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

```
days: Mon
```

```
days: Wed
```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

    - **spec.disruptions.rollingUpdate.windows.from**
      string

      Required value

      Start time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

```
from: '13:00'
```

- **spec.disruptions.rollingUpdate.windows.to**
  string

  Required value

  End time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

  ```
  to: '18:30'
  ```

- **spec.fencing**
  object

  Fencing settings for the node group.

  - **spec.fencing.mode**
    string

    Fencing mode of the node group. Every mode except `Disabled` selects the built-in FencingSLAProfile with the reaction timings of the group:

    - `Disabled`: fencing is off for the node group. A node group without `fencing` is in this mode too.
    - `Critical`: the `critical` profile, reaction in about 12 seconds. Available only in the commercial editions. In Community Edition (CE), Deckhouse does not let you set this mode. A group that already has this mode works as `Normal`.
    - `Normal`: the `normal` profile, reaction in about 30 seconds.
    - `Slow`: the `slow` profile, reaction in about 60 seconds.
    - `Watchdog`: deprecated. This mode works as `Slow`. Use `Normal`, `Slow`, or `Critical` (`Critical` is available only in the commercial editions).

    When at least one node group has a mode other than `Disabled`, the fencing-agent runs on every node of the cluster. If all node groups are `Disabled`, Deckhouse does not deploy the fencing-agent.

    In a cluster with one master, Deckhouse does not let you turn on fencing of an existing node group and shows a warning when you create a node group with fencing, because a failure or a restart of this master stops all nodes with fencing.

    How the fencing-agent works:

    - The agents of all cluster nodes form a single gossip network (`memberlist`). Every agent uses the timings of the fastest mode of the node groups in this network. The memberlist protocol detects failures correctly only when all members have the same timings.
    - On the nodes of a group with a mode other than `Disabled`, the agent arms the `/dev/watchdog` device with the watchdog timeout of the profile of the group. While the agent runs, it resets the timer. If the node or the agent hangs, the timer expires and the kernel panics.
    - When the node does not see the control plane for longer than `controlPlaneLostAfter` of the profile, the agent stops the reset of the timer at once. When the watchdog timeout expires, the node stops. If the control plane is available again before the timeout expires, the agent resets the timer again, and the node continues to work. Thus, a short pause of the control plane does not stop the nodes.
    - After the stop of the reset, the agent asks the agents of the other nodes for a witness, that is, a node that sees the control plane. The witness writes an attestation of the stop. After that, the stop is final.
    - On the nodes of a `Disabled` group, the agent only takes part in failure detection. The agent does not arm the watchdog.
    - The agent respects maintenance annotations `node-manager.deckhouse.io/fencing-disable`, `update.node.deckhouse.io/approved`, and `update.node.deckhouse.io/disruption-approved`, temporarily disabling the watchdog during planned operations.

    For a mode other than `Disabled`, the NodeGroupConfiguration script loads the `softdog` kernel module with `soft_panic=1`. The fencing-agent applies the watchdog timeout when it arms the device.

On nodes of a group with `systemType: Immutable` (Deckhouse Engine), the NodeGroupConfiguration script does not run. On these nodes, `softdog` is built into the kernel and starts with `soft panic=1`. The operating system image sets the permissions on `/dev/watchdog`. The fencing-agent applies the timeout of the profile in the same way as on other nodes. If you installed a node before the Deckhouse Engine release with `softdog` in the kernel, the node does not have `softdog`. The node gets it only after a reinstall or an update of the boot partition.

For a mode other than `Disabled`, Deckhouse disables the automatic reboot of the node after a kernel panic at the OS level. This prevents a node from returning in an undefined state: the stopped node stays down until you restore it manually.

When the stop of the node is certain, the fencing-controller sets the `node.kubernetes.io/out-of-service` taint on the node. After the taint, the controller sets the `Failed` phase and the `DisruptionTarget` condition on the pods of the node and deletes the pods immediately. This includes the pods that tolerate the taint. The controller does not delete the pods of DaemonSets and the static pods. The controller does not remove the finalizers of the pods: a pod with a finalizer stays in the API until the owner of the finalizer removes it. For such a pod of a StatefulSet, the controller sends a warning event with the names of the finalizers. The evacuation is complete when the node has no `VolumeAttachment` objects. If the evacuation is not complete after 60 seconds, the incident stays in the `Evicting` phase and gets the `Progressing` condition with the status `False` and the reason. The controller continues to examine the node. After the evacuation, Deckhouse acts on the node as follows:

- For a node of a `CloudPermanent`, `CloudStatic`, or `Static` group (this includes a `Static` group with `staticInstances`, that is, CAPS), for a master node, and for an etcd arbiter node, Deckhouse only keeps the taint. It never deletes the Node object.
- For a node of a `CloudEphemeral` group with `status.engine: CAPI`, Deckhouse deletes the Node object only when the MachineHealthCheck of the Cluster API cluster reports its machines and 50 percent or fewer of them are unhealthy. Deckhouse counts the stopped machine as unhealthy. MachineHealthCheck then sets the `HealthCheckSucceeded` condition of the Machine to `False` with the `NodeDeleted` reason and replaces the machine. If more than 50 percent of the machines are unhealthy, or the MachineHealthCheck does not report its machines yet, Deckhouse does not delete the Node object yet. The controller continues to examine the machines. When 50 percent or fewer of them are unhealthy, the controller deletes the Node object and completes the evacuation.
- For a node of a `CloudEphemeral` group with `status.engine: MCM`, Deckhouse deletes the `Machine` (`machine.sapcloud.io`). Machine Controller Manager creates a new machine.

When the node returns (the Node is `Ready` and it has a new boot ID), the fencing-controller removes the taint automatically. The controller works only while at least one node group has a mode other than `Disabled`. If you turn fencing off in all node groups, the taint stays on the node, and you must remove it manually.

The stop is certain in one of two cases: a witness confirmed the stop and `watchdog.timeout` + `watchdog.margin` of the profile passed, or the node is silent for `controlPlaneLostAfter` + 0.2s + `watchdog.timeout` + `watchdog.margin` + 0.9s of the profile after its last sign of life. The controller does not set the taint while the `Fencing` condition of the Node is not `True`.

If the fencing-controller does not set the taint, Kubernetes evicts the pods of the node after the time in the `failedNodePodEvictionTimeoutSeconds` parameter of the `control-plane-manager` module (300 seconds by default).

For details on how fencing handles a failed node, see FAQ.

Default: `Disabled`

Allowed values: `Disabled` — Fencing is off for the node group. A node group without fencing is in this mode too.
`Normal` — The normal profile, reaction in about 30 seconds.
`Slow` — The slow profile, reaction in about 60 seconds.
`Critical` — The critical profile, reaction in about 12 seconds. Available only in the commercial editions.
`Watchdog` — Deprecated. Works as Slow. Use Normal, Slow, or Critical (Critical is available only in the commercial editions).

- **spec.fencing.watchdog**
  object

  Deprecated. Deckhouse keeps the settings of fencing 1.0 only for compatibility and

ignores them.

- **spec.fencing.watchdog.timeout**
  string

  Deprecated. Deckhouse ignores this field. The watchdog timeout comes from the profile of the fencing mode.

  Pattern: `^[0-9]+[smh]{1}$`

- **spec.gpu**
  object

  Available in editions: Ultimate/EE

  GPU parameters.

  The section is used by the `gpu` module: it selects the node groups that the module serves in Device Plugin mode. If the `gpu` module is disabled, the section has no effect.

  - **spec.gpu.exclusive**
    object

    Without GPU sharing.

  - **spec.gpu.mig**
    object

    `MIG` sharing strategy parameters.

    - **spec.gpu.mig.customConfigs**
      array of objects

      Custom MIG partitioning by GPU index.

      Used only when `partedConfig: custom` is set.

      Represents an array of devices (a map keyed by `index`), where each entry refers to a GPU by its index and describes the slices to be created.

      Examples:

      ```
      customConfigs:
        profile: 3g.40gb
        count: 2
      ```

      ```
      customConfigs:
        profile: 1g.10gb.me
        count: 1
      ```

      - **spec.gpu.mig.customConfigs.index**
        integer

        Required value

        GPU index on the node.

        Allowed values: `0 <= X`

      - **spec.gpu.mig.customConfigs.slices**
        array of objects

        Required value

        List of MIG slices to create.

        - **spec.gpu.mig.customConfigs.slices.count**
          integer

          Required value

          Number of slices in the selected profile.

          Default: `1`

          Allowed values: `1 <= X <= 7`

- **spec.gpu.mig.customConfigs.slices.profile**
  string

  Required value

  MIG profile name.

  Pattern: `^[0-9]+g\.[0-9]+gb(\.me)?$`

  Examples:

  ```
  profile: 3g.40gb
  ```

  ```
  profile: 1g.10gb
  ```

  ```
  profile: 1g.10gb.me
  ```

- **spec.gpu.mig.partedConfig**
  string

  MIG configuration name.

  Standard `all-*` profiles are supported (the same partitioning applied to all GPUs), as well as the `custom` value, which allows defining partitioning individually for each GPU.

  Default: `all-1g.5gb`

- **spec.gpu.sharing**
  string

  GPU sharing strategy:

  - `Exclusive` — no sharing (each GPU is fully allocated to a single Pod).
  - `TimeSlicing` — time-sliced sharing (multiple Pods can share one GPU concurrently by splitting GPU time).
  - `MIG` — use NVIDIA Multi-Instance GPU (MIG) to split each GPU into multiple instances (if supported by hardware).

  Allowed values: `TimeSlicing`, `MIG`, `Exclusive`

- **spec.gpu.timeSlicing**
  object

  `TimeSlicing` sharing strategy parameters.

  - **spec.gpu.timeSlicing.partitionCount**
    number

    Count of partition per GPU.

    Default: `4`

    Allowed values: `1 <= X <= 100`

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    Default: `4`

    Allowed values: `2 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    Default: `50Mi`

Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.maxPods**
  integer

  Set the max count of pods per node for given NodeGroup. When left unspecified, the system automatically determines this limit according to the value of the `podSubnetNodeCIDRPrefix` parameter of the `control-plane-manager` module:

  - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
  - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
  - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
  - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.
- **spec.kubelet.memorySwap**
  object

  Configuration of swap memory usage for kubelet.

  - **spec.kubelet.memorySwap.limitedSwap**
    object

    Configuration for the limited swap mode. Required when `swapBehavior` is set to `LimitedSwap`.

    - **spec.kubelet.memorySwap.limitedSwap.size**
      string

      Required value

      Size of the swap file to create, specified in GB (for example, “2G”, “4G”).

      Pattern: `^\d+G$`

      Example:

      ```
      size: 1G
      ```
  - **spec.kubelet.memorySwap.swapBehavior**
    string

    Defines how swap memory is handled on the node:

    - `NoSwap`: Swap is disabled. This is the traditional Kubernetes behavior.
    - `LimitedSwap`: Kubelet can use swap memory up to the limit specified in `limitedSwap.size`.

    For details on swap memory usage on nodes, refer to Kubernetes documentation.

    Allowed values: `NoSwap`, `LimitedSwap`
  - **spec.kubelet.memorySwap.swappiness**
    integer

    Defines the kernel’s tendency to use the swap.

    - `0`: Minimal swap usage (only to avoid OOM).
    - `60`: Default balanced behavior.
    - `100`: Aggressive swap usage.

    This value is set via `vm.swappiness` sysctl parameter.

    Default: `60`

    Allowed values: `0 <= X <= 100`
- **spec.kubelet.resourceReservation**
  object

  Management of resource reservation for system daemons on a node.

  More info in the Kubernetes documentation.

  - **spec.kubelet.resourceReservation.mode**
    string

    Defines the system resource reservation mode:

- `Off` — disables resource reservation.
- `Auto` — reserves resources based on the Node capacity.
- `Static` — reserves resources based on the values provided by the user via the `static` parameter.

Note that currently we do not use a dedicated group `cgroup` for resource reservation (`-system-reserved-cgroup` is not used).

Default: `Auto`

- **spec.kubelet.resourceReservation.static**
  object

  Resource reservation parameters for the ‘Static’ mode.

  - **spec.kubelet.resourceReservation.static.cpu**
    integer or string

    Pattern: `\d+[m]`

  - **spec.kubelet.resourceReservation.static.ephemeralStorage**
    integer or string

    Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

  - **spec.kubelet.resourceReservation.static.memory**
    integer or string

    Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.rootDir**
  string

  Directory path for managing kubelet files (volume mounts,etc).

  Default: `/var/lib/kubelet`

- **spec.kubelet.seccompDefault**
  boolean

  Use `RuntimeDefault` seccomp profile for workloads that don’t specify seccomp profile explicitly.

- **spec.kubelet.topologyManager**
  object

  Controls the kubelet resource managers on NUMA nodes: the CPU manager, the memory manager and the topology manager.

  With a policy other than `None`, containers of Guaranteed pods with an integer CPU request get exclusive CPUs, and the memory of Guaranteed pods is bound to NUMA nodes. With `Restricted` and `SingleNumaNode`, they get whole physical cores. Deckhouse calculates NUMA-dependent kubelet settings on each node of the group: it splits `reservedMemory` between NUMA nodes and sets `max-allowable-numa-nodes` on nodes with more than 8 NUMA nodes.

  A policy other than `None` requires `resourceReservation.mode` `Auto`, or `Static` with a positive `static.cpu`. Changing the settings restarts the kubelet and can require a disruption on nodes with Guaranteed pods. See How do I give pods dedicated CPUs and NUMA alignment?

  For more information, see the Kubernetes documentation.

  - **spec.kubelet.topologyManager.enabled**
    Deprecated
    boolean

    Deprecated, has no effect. The resource managers are controlled by `policy` alone.

  - **spec.kubelet.topologyManager.policy**
    string

    Resource alignment policy and the related kubelet settings:

    - `None`: no alignment. The CPU and memory managers are not used.
    - `BestEffort`: Guaranteed pods get exclusive CPUs and NUMA-bound memory. Resources are aligned to NUMA nodes when possible; a pod is

admitted even if they cannot be aligned, but it is rejected if its memory cannot be placed.
- `Restricted`: Guaranteed pods get exclusive whole physical cores and NUMA-bound memory. A pod is rejected if its resources cannot be aligned; when a pod needs several NUMA nodes, the closest ones are preferred.
- `SingleNumaNode`: as `Restricted`, but all resources of a pod (or of a container, see `scope`) must come from one NUMA node.

More information can be found in the Kubernetes documentation.

Default: `None`

Allowed values: `None`, `BestEffort`, `Restricted`, `SingleNumaNode`

- **spec.kubelet.topologyManager.scope**
  string

  Defines granularity with which resources are aligned:

  - `Container`: resources are aligned for each container separately.
  - `Pod`: all containers of a pod are aligned together.

  Default: `Pod`

  Allowed values: `Container`, `Pod`

- **spec.nodeDrainTimeoutSecond**
  integer

  Maximum duration (in seconds) for which the controller attempts to reschedule pods to a new machine during node draining.

  When a machine is deleted, the `Drain` operation evicts the pods for this long. Then the machine is deleted anyway. In a `Static` or `CloudStatic` NodeGroup without this field, the `Drain` of a deleted machine has no time limit.

  The range is the one the module's own settings have always enforced. A value outside it used to be accepted here and then rejected when the module read it, which stopped node-manager reconciling anything at all.

  Allowed values: `30 <= X <= 7200`

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

  ```
  nodeTemplate:
    labels:
      environment: production
      app: warp-drive-ai
    annotations:
      ai.fleet.com/discombobulate: 'true'
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
  ```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **spec.nodeTemplate.labels**
    object

    Similar to the standard `metadata.labels` field.

Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **spec.nodeTemplate.taints**
  array of objects

  Similar to the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **spec.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.nodeTemplate.taints.key**
    string
  - **spec.nodeTemplate.taints.value**
    string

- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides:

  - `CloudEphemeral` — nodes for this group will be automatically created (and deleted) in the cloud of the specified cloud provider;
  - `CloudPermanent`. Nodes are created and deleted using `dhctl`. Virtual machine parameters are set in the cloud provider configuration. When the provider is configured through ModuleConfig, the parameters are set in the instance class referenced by `cloudInstances.classReference`;
  - `CloudStatic` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud providers. This node has the CSI running, and it is managed by the cloud-controller-manager: the `Node` object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes;
  - `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even if one of the cloud providers is enabled.

  Allowed values: `CloudEphemeral`, `CloudPermanent`, `CloudStatic`, `Static`

- **spec.operatingSystem**
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    Deprecated
    boolean

    This parameter has no effect. Earlier, it enabled kernel maintenance on behalf of bashible.

    Default: `true`

- **spec.providerType**
  string

  Cloud provider type of the node group. `None` or an empty value — for a static node group.

- **spec.staticInstances**
  object

  Parameter for provisioning static machines to the cluster.

  - **spec.staticInstances.count**
    integer

    Target number of StaticInstance resources (matching the `labelSelector`) that Deckhouse Platform bootstraps and adds into the cluster.

    In a NodeGroup with Static or CloudStatic nodes, this parameter sets the expected number of nodes. Deckhouse Platform uses this value for state control and automation purposes.

    > **Warning.** Set this parameter explicitly. The default value is `0`, which means Deckhouse Platform won't bootstrap any StaticInstance resource or add any node to the group unless this value is modified. To add static nodes, specify the required number here (greater than `0`).

    Default: `0`

    Allowed values: `0 <= X`

  - **spec.staticInstances.labelSelector**
    object

    Label selector for choosing the StaticInstance resources the node group uses.

    If both `matchLabels` and `matchExpressions` are set, a StaticInstance resource must satisfy all the conditions at once.

    An empty selector (`{}`) chooses every StaticInstance resource. If the selector is not set, the node group chooses no StaticInstance resources.

    Warning. A selector with at least one condition cannot be modified or removed. To change it, create a new node group and move the static nodes to it.

    Adding a selector to a node group that already uses StaticInstance resources is possible only when the labels of all those resources satisfy the new selector. Otherwise the webhook rejects the change and lists the resources that do not match the selector in the error message.

    After the selector is added, existing static nodes are not recreated and keep their previous configuration, but the platform applies the selector to every new node. A StaticInstance resource that does not match the selector cannot be reused, so the node group ends up with fewer nodes than the `count` parameter specifies.

    To add a selector:

    - Label all the StaticInstance resources of the node group with the `d8 k label staticinstance <STATIC INSTANCE> <KEY>=<VALUE>` command.
    - Set the selector in the `spec.staticInstances` parameter of the NodeGroup resource.
    - **spec.staticInstances.labelSelector.matchExpressions**
      array of objects

      A list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **spec.staticInstances.labelSelector.matchExpressions.key**
        string

        key is the label key that the selector applies to.

      - **spec.staticInstances.labelSelector.matchExpressions.operator**
        string

        operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **spec.staticInstances.labelSelector.matchExpressions.values**
        array of strings

values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **Element of the array**
  string

  Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

  Length: `1..63`

- **spec.staticInstances.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **spec.systemType**
  string

  How the nodes of the group are managed.

  - `GenericOS`: Generic OS nodes, configured by bashible on the fly. An unset field means the same.
  - `DeckhouseEngine`: Deckhouse Engine nodes. The node agent brings the node to the state described in a NodeConfig object, without bashible. The machine must boot from a Deckhouse Engine image.

  `DeckhouseEngine` is supported for:

  - `Static`: Machines must be added with `d8 node add`. `staticInstances` cannot be used.
  - `CloudStatic`: Machines must be created manually or using external tools.
  - `CloudEphemeral`: Machines must be created by the cloud provider through Cluster API (CAPI).
  - `CloudPermanent`: Machines are created by dhctl.

  A group can become `DeckhouseEngine` only while bashible has configured none of its nodes. `DeckhouseEngine` cannot be removed or changed to `GenericOS`.

  Allowed values: `GenericOS`, `DeckhouseEngine`

- **spec.update**
  object
  - **spec.update.maxConcurrent**
    integer or string

    Maximum number of concurrently updating nodes.

    Can be set as absolute count or as a percent of total nodes.

    Default: `1`

    Pattern: `^[1-9][0-9]*%?$`

- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

      Allowed values: `True`, `False`

    - **status.conditionSummary.statusMessage**
      string

      Status message about group handling.

  - **status.conditions**
    array of objects
    - **status.conditions.lastTransitionTime**

string

Last time the condition transit from one status to another.

- **status.conditions.message**
  string

  Human readable message indicating details about last transition.

- **status.conditions.status**
  string

  Required value

  Status of the condition, one of True, False.

- **status.conditions.type**
  string

  Required value

  Type of node group condition.

- **status.deckhouse**
  object
  - **status.deckhouse.observed**
    object

    Contains last timestamp when the resource change was noted by the operator and its checksum.

    - **status.deckhouse.observed.checkSum**
      string

      The checksum of the last observed resource.

    - **status.deckhouse.observed.lastTimestamp**
      string

      Last timestamp when a change of the resource was noted by the operator.

  - **status.deckhouse.processed**
    object

    Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

    - **status.deckhouse.processed.checkSum**
      string

      The checksum of the last applied resource.

    - **status.deckhouse.processed.lastTimestamp**
      string

      Last timestamp when the resource was applied in the cluster.

  - **status.deckhouse.synced**
    string

    True if last observed version of the resource was successfully applied in the cluster.

- **status.desired**
  integer

  Number of desired machines in the group.

- **status.engine**
  string

  Machinery used to manage nodes in this group.

  The field value indicates which component manages the nodes:

  - `None` — nodes in the group are not managed by Machine Controller Manager or CAPI.
  - `MCM` — nodes in the group are managed by Machine Controller Manager.
  - `CAPI` — nodes in the group are managed by Cluster API.

  Allowed values: `None`, `MCM`, `CAPI`

- **status.error**
  string

  Error message about possible problems with the group handling.

- **status.instances**
  integer

  Number of instances (in any state) in the group.

- **status.kubernetesVersion**
  string

  The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

- **status.lastMachineFailures**
  array of objects
  - **status.lastMachineFailures.lastOperation**
    object
    - **status.lastMachineFailures.lastOperation.description**
      string

      Last operation’s description.

    - **status.lastMachineFailures.lastOperation.lastUpdateTime**
      string

      Timestamp of last status update for operation.

    - **status.lastMachineFailures.lastOperation.state**
      string

      Machine’s operation state.

    - **status.lastMachineFailures.lastOperation.type**
      string

      Type of operation.

  - **status.lastMachineFailures.name**
    string

    Machine’s name.

  - **status.lastMachineFailures.ownerRef**
    string

    Machine owner’s name.

  - **status.lastMachineFailures.providerID**
    string

    Machine’s ProviderID.

- **status.max**
  integer

  Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

  - **status.upToDate**
    integer

    Number of up-to-date nodes in the group.

Describes the runtime parameters of the node group.

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`
- **spec**
  object

  Required value

  - **spec.chaos**
    object

    Chaos monkey settings.

    Example:

    ```
    chaos:
      mode: DrainAndDelete
      period: 24h
    ```

    - **spec.chaos.mode**
      string

      The chaos monkey mode:

      - `DrainAndDelete` — drains and deletes a node when triggered;
      - `Disabled` — leaves this NodeGroup intact.

      Default: `Disabled`

      Allowed values: `Disabled`, `DrainAndDelete`

    - **spec.chaos.period**
      string

      The time interval to use for the chaos monkey (can be specified in the Go format).

      Default: `6h`

      Pattern: `^[0-9]+[mh]{1}$`
  - **spec.cloudInstances**
    object

    Parameter for provisioning the cloud-based VMs.

    > **i** **Caution.** Can only be used together with `nodeType: CloudEphemeral`.

    - **spec.cloudInstances.classReference**
      object

      Required value

      The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

      - **spec.cloudInstances.classReference.kind**
        string

        The kind of the `InstanceClass` object (e.g., `OpenStackInstanceClass`). Supported kinds are defined by the installed cloud provider modules and specified in their documentation.

    - **spec.cloudInstances.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

- **spec.cloudInstances.maxPerZone**
  integer

  Required value

  The maximum number of instances for the group in each zone.

  This value is used as the upper bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.maxSurgePerZone**
  integer

  The maximum number of instances the platform creates in each zone above the desired number while it replaces the instances of the group.

  Replacement happens when parameters that require recreating a virtual machine change, for example in the InstanceClass. The parameter does not limit the number of instances that the cluster autoscaler orders when it scales the group up. For details on what NodeGroup parameters affect, see NodeGroup parameters and their result.

  The value `0` is treated as if the parameter is not set, so the default value applies.

  Default: `1`

  Allowed values: `0 <= X`

- **spec.cloudInstances.maxUnavailablePerZone**
  integer

  The maximum number of unavailable instances (during rollout) in the group in each zone.

  Default: `0`

  Allowed values: `0 <= X`

- **spec.cloudInstances.minPerZone**
  integer

  Required value

  The minimum number of instances for the group in each zone.

  This value is used in the `MachineDeployment` object and as a lower bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.standby**
  integer or string

  The summary number of overprovisioned nodes for this `NodeGroup` in all zones.

  An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

  The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

  Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
  object

  Amount of reserved resources.

  Used to determine whether to order overprovisioned nodes.

- **spec.cloudInstances.standbyHolder.notHeldResources**
  object

  Describes the resources that will not be held (consumed) by the standby holder.

  - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
    integer or string

    Describes the amount of CPU that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

    Pattern: `^[0-9]+m?$`

  - **spec.cloudInstances.standbyHolder.notHeldResources.memory**
    integer or string

    Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.zones**
  array of strings

  List of availability zones to create instances in.

  The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

  Example:

  ```
  zones:
  - Helsinki
  - Espoo
  - Tampere
  ```

- **spec.cri**
  object

  Container runtime parameters.

  - **spec.cri.containerd**
    object

    Containerd runtime parameters.

    If used, `cri.type` must be set to `Containerd`.

    - **spec.cri.containerd.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

      Default: `3`

  - **spec.cri.docker**
    object

    Docker settings for nodes.

    > **Note.** the `Docker` is deprecated.

    - **spec.cri.docker.manage**
      boolean

      Enable Docker maintenance from bashible.

      Default: `true`

        - **spec.cri.docker.maxConcurrentDownloads**
          integer

          Set the max concurrent downloads for each pull.

          Default: `3`

      - **spec.cri.notManaged**
        object

        Settings for not managed CRI for nodes.

        - **spec.cri.notManaged.criSocketPath**
          string

          Path to CRI socket.

      - **spec.cri.type**
        string

        Container runtime type.

        Value `defaultCRI` from the initial cluster configration (`cluster-configuration.yaml` parameter from the `d8-cluster-configuration` secret in the `kube-system` namespace) is used if not specified.

        > **i** **Note.** Docker support is deprecated.

        Allowed values: `Docker`, `Containerd`, `NotManaged`

  - **spec.disruptions**
    object

    Disruptions settings for nodes.

    Example:

    ```
    disruptions:
      approvalMode: Automatic
      automatic:
        drainBeforeApproval: false
        windows:
        - from: '06:00'
          to: '08:00'
          days:
          - Tue
          - Sun
    ```

    - **spec.disruptions.approvalMode**
      string

      The approval mode for disruptive updates:

      - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. **Caution.** The `master` node group update mode must be `Manual` to avoid issues with draining.
      - `Automatic` — automatically approve disruption-involving updates.
      - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

      If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

      Default: `Automatic`

      Allowed values: `Manual`, `Automatic`, `RollingUpdate`

- **spec.disruptions.automatic**
  object

  Additional parameters for the `Automatic` mode.

  - **spec.disruptions.automatic.drainBeforeApproval**
    boolean

    Drain Pods from the nodes before approving disruption.

    Caution. This setting ignores (nodes will be approved without draining Pods):

    - for the nodeGroup `master` with a single node;
    - for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

    Default: `true`

  - **spec.disruptions.automatic.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.automatic.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

    - **spec.disruptions.automatic.windows.from**
      string

      Required value

      Start time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      from: '13:00'
      ```

    - **spec.disruptions.automatic.windows.to**
      string

      Required value

      End time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      to: '18:30'
      ```

- **spec.disruptions.rollingUpdate**
  object

  Additional parameters for the `RollingUpdate` mode.

  - **spec.disruptions.rollingUpdate.windows**
    array of objects

    Time windows for node disruptive updates

Time windows for node disruptive updates.

- **spec.disruptions.rollingUpdate.windows.days**
  array of strings

  Days of the week when node could be updated.

  Examples:

  ```
  days: Mon
  ```

  ```
  days: Wed
  ```

  - **Element of the array**
    string

    Day of the week.

    Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

- **spec.disruptions.rollingUpdate.windows.from**
  string

  Required value

  Start time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

  ```
  from: '13:00'
  ```

- **spec.disruptions.rollingUpdate.windows.to**
  string

  Required value

  End time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

  ```
  to: '18:30'
  ```

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    > **Warning.** This parameter does nothing if CRI type is `Docker`.

    Default: `4`

    Allowed values: `2 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    > **Warning.** This parameter does nothing if CRI type is `Docker`.

    Default: `50Mi`

Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.maxPods**
  integer

  Set the max count of pods per node for given NodeGroup. When left unspecified, the system automatically determines this limit according to the value of the `podSubnetNodeCIDRPrefix` parameter of the `control-plane-manager` module:

  - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
  - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
  - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
  - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.
- **spec.kubelet.rootDir**
  string

  Directory path for managing kubelet files (volume mounts,etc).

  Default: `/var/lib/kubelet`

- **spec.kubelet.seccompDefault**
  boolean

  Use `RuntimeDefault` seccomp profile for workloads that don’t specify seccomp profile explicitly.

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

  ```
  nodeTemplate:
    labels:
      environment: production
      app: warp-drive-ai
    annotations:
      ai.fleet.com/discombobulate: 'true'
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
  ```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **spec.nodeTemplate.labels**
    object

    Similar to the standard `metadata.labels` field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **spec.nodeTemplate.taints**
    array of objects

    Similar to the `.spec.taints` field of the Node object.

> **i** Available fields: `effect`, `key`, and `values`.

Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **spec.nodeTemplate.taints.effect**
  string

  Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
- **spec.nodeTemplate.taints.key**
  string
- **spec.nodeTemplate.taints.value**
  string

- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides.

  - `Cloud` — nodes for this group will be automatically created (and deleted) in the cloud of the specified cloud provider;
  - `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even of one of the cloud providers is enabled;
  - `Hybrid` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud provider. This node has the CSI running, and it is managed by the cloud-controller-manager: the Node object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes.

  Allowed values: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    boolean

    Enable kernel maintenance from bashible.

    Default: `true`

- **spec.providerType**
  string

  Cloud provider type of the node group. `None` or an empty value — for a static node group.

- **spec.systemType**
  string

  How the nodes of the group are managed.

  - `GenericOS`: Generic OS nodes, configured by bashible on the fly. An unset field means the same.
  - `DeckhouseEngine`: Deckhouse Engine nodes. The node agent brings the node to the state described in a NodeConfig object, without bashible. The machine must boot from a Deckhouse Engine image.

  `DeckhouseEngine` is supported for:

  - `Static`: Machines must be added with `d8 node add`.
  - `Cloud`: Machines must be created by the cloud provider through Cluster API (CAPI).
  - `Hybrid`: Machines must be created by dhctl (`CloudPermanent` in `deckhouse.io/v1`).

  `DeckhouseEngine` is not supported for other `Hybrid` groups (`CloudStatic` in

`deckhouse.io/v1`).

A group can become `DeckhouseEngine` only while bashible has configured none of its nodes. `DeckhouseEngine` cannot be removed or changed to `GenericOS`.

Allowed values: `GenericOS`, `DeckhouseEngine`

- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

      Allowed values: `True`, `False`

    - **status.conditionSummary.statusMessage**
      string

      Status message about group handling.

  - **status.desired**
    integer

    Number of desired machines in the group.

  - **status.engine**
    string

    Machinery used to manage nodes in this group.

    The field value indicates which component manages the nodes:

    - `None` — nodes in the group are not managed by MCM or CAPI.
    - `MCM` — nodes in the group are managed by machine-controller-manager.
    - `CAPI` — nodes in the group are managed by Cluster API.

    Allowed values: `None`, `MCM`, `CAPI`

  - **status.error**
    string

    Error message about possible problems with the group handling.

  - **status.instances**
    integer

    Number of instances (in any state) in the group.

  - **status.kubernetesVersion**
    string

    The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

  - **status.lastMachineFailures**
    array of objects
    - **status.lastMachineFailures.lastOperation**
      object
      - **status.lastMachineFailures.lastOperation.description**
        string

        Last operation’s description.

      - **status.lastMachineFailures.lastOperation.lastUpdateTime**
        string

        Timestamp of last status update for operation.

      - **status.lastMachineFailures.lastOperation.state**
        string

        Machine’s operation state.

      - **status.lastMachineFailures.lastOperation.type**
        string

Type of operation.

- **status.lastMachineFailures.name**
  string

  Machine’s name.

- **status.lastMachineFailures.ownerRef**
  string

  Machine owner’s name.

- **status.lastMachineFailures.providerID**
  string

  Machine’s ProviderID.

- **status.max**
  integer

  Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

- **status.upToDate**
  integer

  Number of up-to-date nodes in the group.

Defines the runtime parameters of a node group.

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`

- **spec**
  object

  Required value

  - **spec.chaos**
    object

    Chaos monkey settings.

    Example:

    ```
    chaos:
      mode: DrainAndDelete
      period: 24h
    ```

    - **spec.chaos.mode**
      string

      The chaos monkey mode:

The chaos monkey mode:

- `DrainAndDelete` — drains and deletes a node when triggered;
- `Disabled` — leaves this NodeGroup intact.

Default: `Disabled`

Allowed values: `Disabled`, `DrainAndDelete`

- **spec.chaos.period**
  string

  The time interval to use for the chaos monkey (can be specified in the Go format).

  Default: `6h`

  Pattern: `^[0-9]+[mh]{1}$`

- **spec.cloudInstances**
  object

  Parameter for provisioning the cloud-based VMs.

  > **Caution.** Can only be used together with `nodeType: CloudEphemeral`.

  - **spec.cloudInstances.classReference**
    object

    Required value

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **spec.cloudInstances.classReference.kind**
      string

      The kind of the `InstanceClass` object (e.g., `OpenStackInstanceClass`). Supported kinds are defined by the installed cloud provider modules and specified in their documentation.

    - **spec.cloudInstances.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **spec.cloudInstances.maxPerZone**
    integer

    Required value

    The maximum number of instances for the group in each zone.

    This value is used as the upper bound in cluster-autoscaler.

    With a value of 0, you need to set capacity for some `InstanceClass`. Get more details in the description of the necessary `InstanceClass`.

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxSurgePerZone**
    integer

    The maximum number of instances the platform creates in each zone above the desired number while it replaces the instances of the group.

    Replacement happens when parameters that require recreating a virtual machine change, for example in the InstanceClass. The parameter does not limit the number of instances that the cluster autoscaler orders when it scales the group up. For details on what NodeGroup parameters affect, see NodeGroup parameters and their result.

    The value `0` is treated as if the parameter is not set, so the default value applies.

    Default: `1`

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxUnavailablePerZone**

integer

The maximum number of unavailable instances (during rollout) in the group in each zone.

Default: `0`

Allowed values: `0 <= X`

- **spec.cloudInstances.minPerZone**
  integer

  Required value

  The minimum number of instances for the group in each zone.

  This value is used in the `MachineDeployment` object and as a lower bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.standby**
  integer or string

  The summary number of overprovisioned nodes for this `NodeGroup` all zones.

  An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

  The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

  Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
  object

  Amount of reserved resources.

  Used to determine whether to order overprovisioned nodes.

  - **spec.cloudInstances.standbyHolder.notHeldResources**
    object

    Describes the resources that will not be held (consumed) by the standby holder.

    - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
      integer or string

      Describes the amount of CPU that will not be held by standby holder on Nodes from this NodeGroup.

      The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

      Pattern: `^[0-9]+m?$`

    - **spec.cloudInstances.standbyHolder.notHeldResources.memory**
      integer or string

      Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

      The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.zones**
  array of strings

  List of availability zones to create instances in.

  The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

Example:

```
zones:
- Helsinki
- Espoo
- Tampere
```

- **spec.cri**
  object

  Container runtime parameters.

  - **spec.cri.containerd**
    object

    Containerd runtime parameters.

    If used, `cri.type` must be set to `Containerd`.

    - **spec.cri.containerd.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

      Default: `3`

  - **spec.cri.type**
    string

    Container runtime type.

    If not specified, the `defaultCRI` parameter value of ClusterConfiguration is used.

    > **Note.** Docker support is deprecated.

    Allowed values: `Docker`, `Containerd`, `NotManaged`

- **spec.disruptions**
  object

  Disruptions settings for nodes.

  Example:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
    windows:
    - from: '06:00'
      to: '08:00'
      days:
      - Tue
      - Sun
```

  - **spec.disruptions.approvalMode**
    string

    The approval mode for disruptive updates:

    - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. **Caution.** The `master` node group update mode must be `Manual` to avoid issues with draining.
    - `Automatic` — automatically approve disruption-involving updates.
    - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

    If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the

node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

Default: `Automatic`

Allowed values: `Manual`, `Automatic`, `RollingUpdate`

- **spec.disruptions.automatic**
  object

  Additional parameters for the `Automatic` mode.

  - **spec.disruptions.automatic.drainBeforeApproval**
    boolean

    Drain Pods from the nodes before approving disruption.

    Caution. This setting ignores (nodes will be approved without draining Pods):

    - for the nodeGroup `master` with a single node;
    - for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

    Default: `true`

  - **spec.disruptions.automatic.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.automatic.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

    - **spec.disruptions.automatic.windows.from**
      string

      Required value

      Start time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      from: '13:00'
      ```

    - **spec.disruptions.automatic.windows.to**
      string

      Required value

      End time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

```
to: '18:30'
```

- **spec.disruptions.rollingUpdate**
  object

  Additional parameters for the `RollingUpdate` mode.

  - **spec.disruptions.rollingUpdate.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.rollingUpdate.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

    - **spec.disruptions.rollingUpdate.windows.from**
      string

      Required value

      Start time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      from: '13:00'
      ```

    - **spec.disruptions.rollingUpdate.windows.to**
      string

      Required value

      End time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      to: '18:30'
      ```

- **spec.docker**
  object

  Docker settings for nodes.

  If used, `cri.type` must be set to `Docker`.

  > **i** **Note.** the `Docker` is deprecated.

  - **spec.docker.manage**
    boolean

    Enable Docker maintenance from bashible.

    Default: `true`

- **spec.docker.maxConcurrentDownloads**
  integer

  Set the max concurrent downloads for each pull.

  Default: `3`

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    > **Warning.** This parameter does nothing if CRI type is `Docker`.

    Default: `4`

    Allowed values: `2 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    > **Warning.** This parameter does nothing if CRI type is `Docker`.

    Default: `50Mi`

    Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

  - **spec.kubelet.maxPods**
    integer

    Set the max count of pods per node for given NodeGroup. When left unspecified, the system automatically determines this limit according to the value of the `podSubnetNodeCIDRPrefix` parameter of the `control-plane-manager` module:

    - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
    - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
    - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
    - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.

  - **spec.kubelet.rootDir**
    string

    Directory path for managing kubelet files (volume mounts,etc).

    Default: `/var/lib/kubelet`

  - **spec.kubelet.seccompDefault**
    boolean

    Use `RuntimeDefault` seccomp profile for workloads that don’t specify seccomp profile explicitly.

- **spec.kubernetesVersion**
  string

  The desired minor version of Kubernetes.

  By default, it corresponds to the version selected for the cluster globally (see installation documentation) or to the current version of the control plane (if the global version is not defined).

  Allowed values: `1.33`, `1.34`, `1.35`, `1.36`, `1.37`

  Example:

  ```
  kubernetesVersion: '1.33'
  ```

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

  ```
  nodeTemplate:
    labels:
      environment: production
      app: warp-drive-ai
    annotations:
      ai.fleet.com/discombobulate: 'true'
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
  ```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **spec.nodeTemplate.labels**
    object

    Similar to the standard `metadata.labels` field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **spec.nodeTemplate.taints**
    array of objects

    Similar to the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **spec.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **spec.nodeTemplate.taints.key**
      string
    - **spec.nodeTemplate.taints.value**
      string
- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides.

- `Cloud` — nodes for this group will be automatically created (and deleted) in the cloud of the specified cloud provider;
- `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even of one of the cloud providers is enabled;
- `Hybrid` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud provider. This node has the CSI running, and it is managed by the cloud-controller-manager: the Node object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes.

Allowed values: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**
  Deprecated
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    Deprecated
    boolean

    Enable kernel maintenance from bashible.

    Default: `true`

- **spec.providerType**
  string

  Cloud provider type of the node group. `None` or an empty value — for a static node group.

- **spec.static**
  object

  Static node parameters

  - **spec.static.internalNetworkCIDRs**
    array of strings

    Subnet CIDR

- **spec.systemType**
  string

  How the nodes of the group are managed.

  - `GenericOS`: Generic OS nodes, configured by bashible on the fly. An unset field means the same.
  - `DeckhouseEngine`: Deckhouse Engine nodes. The node agent brings the node to the state described in a NodeConfig object, without bashible. The machine must boot from a Deckhouse Engine image.

  `DeckhouseEngine` is supported for:

  - `Static`: Machines must be added with `d8 node add`.
  - `Cloud`: Machines must be created by the cloud provider through Cluster API (CAPI).
  - `Hybrid`: Machines must be created by dhctl (`CloudPermanent` in `deckhouse.io/v1`).

  `DeckhouseEngine` is not supported for other `Hybrid` groups (`CloudStatic` in `deckhouse.io/v1`).

  A group can become `DeckhouseEngine` only while bashible has configured none of its nodes. `DeckhouseEngine` cannot be removed or changed to `GenericOS`.

  Allowed values: `GenericOS`, `DeckhouseEngine`

- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

      Allowed values: `True`, `False`

        - **status.conditionSummary.statusMessage**
          string

          Status message about group handling.

    - **status.desired**
      integer

      Number of desired machines in the group.

    - **status.engine**
      string

      Machinery used to manage nodes in this group.

      The field value indicates which component manages the nodes:

      - `None` — nodes in the group are not managed by Machine Controller Manager or CAPI.
      - `MCM` — nodes in the group are managed by Machine Controller Manager.
      - `CAPI` — nodes in the group are managed by Cluster API.

      Allowed values: `None`, `MCM`, `CAPI`

    - **status.error**
      string

      Error message about possible problems with the group handling.

    - **status.instances**
      integer

      Number of instances (in any state) in the group.

    - **status.kubernetesVersion**
      string

      The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

    - **status.lastMachineFailures**
      array of objects
      - **status.lastMachineFailures.lastOperation**
        object
        - **status.lastMachineFailures.lastOperation.description**
          string

          Last operation’s description.

        - **status.lastMachineFailures.lastOperation.lastUpdateTime**
          string

          Timestamp of last status update for operation.

        - **status.lastMachineFailures.lastOperation.state**
          string

          Machine’s operation state.

        - **status.lastMachineFailures.lastOperation.type**
          string

          Type of operation.

      - **status.lastMachineFailures.name**
        string

        Machine’s name.

      - **status.lastMachineFailures.ownerRef**
        string

        Machine owner’s name.

      - **status.lastMachineFailures.providerID**
        string

        Machine’s ProviderID.

    - **status.max**

integer

Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

- **status.upToDate**
  integer

  Number of up-to-date nodes in the group.

NodeConfig

Short names: nc

Scope: Cluster
Version: v1alpha1

NodeConfig is the desired state of one node and the node agent’s report on it.

The node agent (nodelet) applies `spec`.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  Required value

  Desired state of the node.

  - **spec.apiServerEndpoints**
    array of strings

    Kubernetes API server URLs with scheme and port, for example `https://10.0.0.1:6443`.

    - **Element of the array**
      string

      Pattern: `^https://(\[[0-9A-Fa-f:]+\]|[A-Za-z0-9]([-A-Za-z0-9]*[A-Za-z0-9])?([.][A-Za-z0-9]([-A-Za-z0-9]*[A-Za-z0-9])?)*):(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|6[0-4][0-9]{3}|[1-5][0-9]{4}|[1-9][0-9]{0,3})/?$`

      Maximum length: `253`

- **spec.containerRuntime**
  object

  ContainerRuntime holds containerd configuration.

  - **spec.containerRuntime.maxConcurrentDownloads**
    integer

    Maximum number of image layers downloaded in parallel.

    `0` means no limit. Defaults to `8` when unset.

    Allowed values: `0 <= X`

  - **spec.containerRuntime.registryConfigOwner**
    string

    Component that writes the containerd `registry.d` directory.

    - `Nodelet`: The node agent writes the configuration from `spec.registry` to the directory and removes everything else in it, except the route directories of `spec.registry.upstreams`.
    - `RegistryAgent`: The registry agent static pod owns the directory. The node agent writes nothing there and only removes the directories it wrote before.

    Defaults to `Nodelet` when unset. Set by `node-manager`.

    Allowed values: `Nodelet`, `RegistryAgent`

  - **spec.containerRuntime.sandboxImage**
    string

    The `pause` image used for pod sandboxes.

    If not specified, the pause image imported by the containerd system extension is used.

    Pattern: `^[^[:space:]]+$`

    Maximum length: `1024`

- **spec.extensions**
  array of objects

  Extensions the node installs, each pinned by digest.

  On a Deckhouse Engine node an entry is a signed dm-verity system extension image. On a Generic OS node an entry is a registry package: the node agent runs the package's own `install` script as root.

  An entry named `nodelet` updates the node agent itself.

  Extension is a signed verity sysext built from a release channel, fetched from the registry-packages-proxy by digest. Optional repository selects the proxy's per-registry config; optional path is its "path" parameter.

  - **spec.extensions.digest**
    string

    Required value

    Digest is the sha256 digest of the sysext image.

    Pattern: `^sha256:[a-f0-9]{64}$`

  - **spec.extensions.name**
    string

    Required value

    Name is the extension name (also the sysext image basename).

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Length: `1..63`

  - **spec.extensions.path**
    string

    Path to the artifact inside the registry.

Maximum length: `512`

- **spec.extensions.repository**
  string

  Repository optionally selects the proxy’s per-registry client config.

  Pattern: `^[^[:space:]]+$`

  Maximum length: `1024`

- **spec.extensions.requestedBy**
  object

  Object that requested the extension.

  Unset for an extension of the platform itself.

  - **spec.extensions.requestedBy.kind**
    string

    Required value

    Kind of the object.

    Allowed values: `NodeExtensionRequest`, `NodeStaticPodRequest`, `RegistryUpstream`, `ModuleSource`

  - **spec.extensions.requestedBy.name**
    string

    Required value

    Name of the object.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Length: `1..253`

- **spec.featureGates**
  object

  Node agent features to switch on or off, keyed by feature name.

  A feature missing from this map is on for a Deckhouse Engine node if the node type supports it, and off for a Generic OS node. A Generic OS node runs only the features switched on here.

  The node rejects a document that names an unknown feature or switches on a feature its node type does not support.

  Set by node-controller.

- **spec.internalNetworkCIDRs**
  array of strings

  InternalNetworkCIDRs are the subnets the cluster’s own traffic runs on. The node picks the interface holding an address inside one of them and registers with that address; a node with one NIC needs none of this. IPv4 only.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

    Maximum length: `43`

- **spec.kernel**
  object

  Kernel holds sysctl settings and kernel modules to load.

  - **spec.kernel.modules**
    array of objects

    Kernel modules to load, in this order.

    Dependencies are not loaded automatically: list each module after the modules it

depends on. Do not list a driver built into the kernel: `KernelModulesLoaded` becomes `False`.

A module added to the list is loaded only after the disruption is approved.

KernelModule is a module to load with optional parameters.

- **spec.kernel.modules.name**
  string

  Required value

  Module name, as `lsmod` lists it. A `-` and a `_` in the name are the same.

  Length: `1..64`

- **spec.kernel.modules.params**
  array of strings

  Module parameters, passed to `insmod` in this order, for example `debug=1`.

  A changed list applies the next time the module loads: a loaded module is not reloaded.

  - **Element of the array**
    string

    Maximum length: `256`

- **spec.kernel.sysctl**
  object

  Kernel parameters the node sets at runtime, keyed by their dotted name, for example `net.ipv4.ip_forward`.

  A key the kernel does not have is skipped. The keys the node agent requires keep its values, and `vm.overcommit_memory` can only be `1`. The size of the connection tracking table, which the node computes from its cores, yields to a value set here.

- **spec.kubelet**
  object

  Kubelet holds kubelet configuration parameters.

  - **spec.kubelet.bootstrapToken**
    string

    BootstrapToken is the bootstrap token used by kubelet to obtain its client certificate on first boot.

    Maximum length: `4096`

  - **spec.kubelet.clusterCABundle**
    string

    PEM bundle of the cluster CA.

    Maximum length: `65536`

  - **spec.kubelet.clusterDNS**
    array of strings

    ClusterDNS is the list of DNS server IP addresses for the cluster.

    - **Element of the array**
      string

      Maximum length: `45`

  - **spec.kubelet.clusterDomain**
    string

    ClusterDomain is the DNS domain for this cluster (e.g. “cluster.local”).

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Maximum length: `253`

  - **spec.kubelet.containerLogMaxFiles**

integer

Number of rotated log files kept per container.

Defaults to `4` when unset.

Allowed values: `2 <= X <= 1000`

- **spec.kubelet.containerLogMaxSize**
  string

  Maximum size of a container log file before it is rotated.

  Defaults to `50Mi` when unset.

  Maximum length: `64`

- **spec.kubelet.externalCloudProvider**
  boolean

  ExternalCloudProvider enables –cloud-provider=external so the cloud-controller-manager manages the node (zone, region, providerID).

- **spec.kubelet.kubernetesVersion**
  string

  KubernetesVersion is the cluster’s minor version, e.g. “1.34”. It decides which feature gates kubelet is started with (bashible turns DRA gates on by version); without it DRA workloads cannot run on Deckhouse Engine nodes.

  Pattern: `^[0-9]+\.[0-9]+$`

  Maximum length: `16`

- **spec.kubelet.maxPods**
  integer

  Maximum number of pods on the node.

  Defaults to `120` when unset.

  Allowed values: `1 <= X <= 1000`

- **spec.kubelet.nodeLabels**
  object

  NodeLabels are labels added to the node object when kubelet registers. A NodeGroup caps neither their number nor their shape, so neither does this.

- **spec.kubelet.registerWithTaints**
  array of objects

  RegisterWithTaints is a list of taints to add to the node object when kubelet registers itself. Only takes effect on initial registration.

  Taint represents a Kubernetes taint applied to a node during registration. Key, value and effect take what a NodeGroup takes: bare strings and an optional effect, so a taint copied from one still fits.

  - **spec.kubelet.registerWithTaints.effect**
    string

    Taint effect. A taint without an effect is skipped: kubelet registers a taint only with one of these.

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.kubelet.registerWithTaints.key**
    string

    Required value

    Taint key. A taint kubelet cannot register, such as one whose key is not a qualified name, is skipped, and the other taints are registered.

  - **spec.kubelet.registerWithTaints.value**
    string

    Taint value.

- **spec.kubelet.resourceReservation**
  object

  ResourceReservation controls how much CPU, memory and disk are held back from pods for the system itself (systemReserved).

  - **spec.kubelet.resourceReservation.mode**
    string

    Reservation policy.

    - `Auto`: Computed from the node's capacity.
    - `Off`: Nothing is reserved.

    Defaults to `Auto` when unset.

    Allowed values: `Auto`, `Off`

- **spec.kubelet.serverTLSBootstrap**
  boolean

  ServerTLSBootstrap makes kubelet request a serving certificate from the cluster instead of signing its own.

  Defaults to `true` when unset.

- **spec.network**
  object

  Network holds hostname, DNS, NTP, interfaces and routes.

  - **spec.network.dns**
    object

    DNS resolver configuration.

    - **spec.network.dns.search**
      array of strings

      Search domains for host names that are not fully qualified, tried in this order.

      - **Element of the array**
        string

        Maximum length: `253`

    - **spec.network.dns.servers**
      array of strings

      Name server IP addresses.

      - **Element of the array**
        string

        Maximum length: `45`

  - **spec.network.hostname**
    string

    Host name of the node.

    Maximum length: `253`

  - **spec.network.interfaces**
    array of objects

    Network interfaces, keyed by name.

    NetworkInterface describes a single NIC.

    - **spec.network.interfaces.addresses**
      array of strings

      Static addresses in CIDR notation, used when `dhcp` is `false`.

      - **Element of the array**
        string

        Maximum length: `43`

- **spec.network.interfaces.cluster**
  boolean

  Cluster names this NIC the one the cluster’s traffic runs on, for a machine whose addressing spec.internalNetworkCIDRs cannot tell apart.

- **spec.network.interfaces.dhcp**
  boolean

  Required value

  DHCP enables DHCPv4 on the interface.

- **spec.network.interfaces.gateway**
  string

  Default gateway IP address.

  Maximum length: `45`

- **spec.network.interfaces.name**
  string

  Required value

  Interface name, for example `eth0`.

  Length: `1..253`

- **spec.network.ntp**
  object

  NTP time-sync configuration.

  - **spec.network.ntp.servers**
    array of strings

    NTP servers, each a host name or an IPv4 address. On Immutable nodes the time-sync feature checks them once per boot, before kubelet starts, and waits at most 30 seconds for a usable answer. The deckhouse chrony module keeps the clock in time after that.

    - **Element of the array**
      string

      Maximum length: `253`

- **spec.network.routes**
  array of objects

  Static routes, added on every interface listed in `interfaces`.

  Route is a static route.

  - **spec.network.routes.gateway**
    string

    Gateway IP address.

    Maximum length: `45`

  - **spec.network.routes.name**
    string

    Label for the route. It does not change how the route is applied.

    Maximum length: `253`

  - **spec.network.routes.networks**
    array of strings

    Destination networks in CIDR notation.

    - **Element of the array**
      string

      Maximum length: `43`

- **spec.nodeName**

string

Required value

Name of the Kubernetes node this configuration applies to.

Equals `metadata.name` and cannot be changed.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Length: `1..253`

- **spec.osImage**
  object

  Operating system image to use on the node.

  To determine whether an update is required, the node compares the image `digest` with the value saved to `/config/os-digest` during installation. The digest identifies a specific image publication and may change when the same content is republished.

  Required when `systemType` is `DeckhouseEngine` or its former name `Immutable`.

  - **spec.osImage.digest**
    string

    Required value

    Operating system image digest.

    Identifies a specific image and is used by the node to determine whether an update is required. During installation, the node saves the digest and compares it with the specified value during subsequent checks.

    Pattern: `^sha256:[a-f0-9]{64}$`

  - **spec.osImage.path**
    string

    Path to the image inside the registry.

    Maximum length: `512`

  - **spec.osImage.repository**
    string

    Address of the container registry from which the operating system image is downloaded.

    The value is passed to `registry-packages-proxy` as the `repository` parameter.

    If the parameter is not specified, `repository` is not passed and `registry-packages-proxy` uses its default container registry.

    Pattern: `^[^[:space:]]+$`

    Maximum length: `1024`

  - **spec.osImage.rootHash**
    string

    Dm-verity root hash of the root filesystem inside the image, in ASCII hexadecimal format.

    Used to determine whether the actual image content has changed. Unlike `digest`, which may change when an image is republished, `rootHash` remains the same if the root filesystem content has not changed.

    Deckhouse Platform obtains the value from an image label. If the value cannot be obtained from the label, the node determines the root hash after downloading the image.

    Pattern: `^[a-f0-9]{64}$`

- **spec.registry**
  object

  Registry is the container registry the node talks to directly, without the registry-packages-proxy. A node bootstrapping a control plane has no proxy yet but must pull images and

sysexts. `upstreams` lists the additional registries containerd routes pulls to.

On a Generic OS node bashible configures containerd for the registry itself, and the node agent writes only the `upstreams` routes, with the `registry-routes` feature on. It reads only `address` there, which no route may take over, and never downloads from the registry directly.

- **spec.registry.address**
  string

  Required value

  Address is the registry host, optionally with a port, e.g. “registry.deckhouse.io” or “registry.example.com:5000”.

  Pattern: `^[^[:space:]]+$`

  Length: `1..1024`

- **spec.registry.auth**
  string

  Auth is the base64-encoded “user:password” pair, as it appears in the “auth” field of a docker config.

  Maximum length: `4096`

- **spec.registry.caBundle**
  string

  PEM bundle of the CA certificates the registry is verified with.

  Maximum length: `65536`

- **spec.registry.path**
  string

  Path is the repository path within the registry, e.g. “/deckhouse/ce”.

  Maximum length: `1024`

- **spec.registry.scheme**
  string

  Protocol the node uses to connect to the registry.

  Defaults to `HTTPS` when unset.

  Allowed values: `HTTPS`, `HTTP`

- **spec.registry.upstreams**
  array of objects

  Additional registries the node routes pulls to: each entry sends the pulls of one registry host to another registry and its mirrors.

  The node writes one containerd host directory per entry, named after `match` in lower case, and removes the directories of entries that are gone. A route change does not restart containerd. An entry the node cannot apply is skipped and reported in the `RegistryRoutesReady` condition; the other entries are still applied.

  The `registry-routes` feature applies this list. It is on by default on a Deckhouse Engine node; a Generic OS node applies the list only when `featureGates` switch the feature on. With the feature off, the node leaves the directories it wrote before as they are.

  Set by node-controller from the accepted RegistryUpstream objects and the CAs of ModuleSource objects. On a Deckhouse Engine node the list is empty while the registry module’s agent configures containerd. On a Generic OS node the list is always set but has no ModuleSource entry, since bashible writes those CAs itself, and the node writes no route while the agent’s `default` directory is listed in bashible’s `deckhouse_hosts_state.json`.

  RegistryUpstream is one route of spec.registry.upstreams.

  - **spec.registry.upstreams.match**
    string

Required value

Registry host, optionally with a port, whose pulls the route takes over. The name of the containerd host directory.

Pattern: `^[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?(\.[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?)*(:[0-9]{1,5})?$`

Length: `1..255`

- **spec.registry.upstreams.requestedBy**
  object

  Required value

  Object the route comes from: a RegistryUpstream, or a ModuleSource whose registry needs its own CA.

  - **spec.registry.upstreams.requestedBy.kind**
    string

    Required value

    Kind of the object.

    Allowed values: `NodeExtensionRequest`, `NodeStaticPodRequest`, `RegistryUpstream`, `ModuleSource`

  - **spec.registry.upstreams.requestedBy.name**
    string

    Required value

    Name of the object.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Length: `1..253`

- **spec.registry.upstreams.upstream**
  object

  Required value

  Registry the pulls are sent to, and its mirrors.

  - **spec.registry.upstreams.upstream.auth**
    object

    Credentials for the registry.

    - **spec.registry.upstreams.upstream.auth.auth**
      string

      Base64-encoded `username:password` pair. Takes precedence over `username` and `password` when set.

      Maximum length: `4096`

    - **spec.registry.upstreams.upstream.auth.password**
      string

      Password for basic authentication.

      Maximum length: `4096`

    - **spec.registry.upstreams.upstream.auth.username**
      string

      Username for basic authentication.

      Maximum length: `1024`

  - **spec.registry.upstreams.upstream.ca**
    string

    PEM bundle of the CA certificates the registry is verified with over `HTTPS`. The node skips the route unless the bundle is only CERTIFICATE blocks that

all parse.

Maximum length: `65536`

- **spec.registry.upstreams.upstream.host**
  string

  Required value

  Registry host, optionally with a port.

  Length: `1..255`

- **spec.registry.upstreams.upstream.mirrors**
  array of objects

  Additional addresses serving the same content, tried after the registry in the listed order.

  RegistryEndpoint is one address of a route.

  - **spec.registry.upstreams.upstream.mirrors.auth**
    object

    Credentials for the registry.

    - **spec.registry.upstreams.upstream.mirrors.auth.auth**
      string

      Base64-encoded `username:password` pair. Takes precedence over `username` and `password` when set.

      Maximum length: `4096`

    - **spec.registry.upstreams.upstream.mirrors.auth.password**
      string

      Password for basic authentication.

      Maximum length: `4096`

    - **spec.registry.upstreams.upstream.mirrors.auth.username**
      string

      Username for basic authentication.

      Maximum length: `1024`

  - **spec.registry.upstreams.upstream.mirrors.ca**
    string

    PEM bundle of the CA certificates the registry is verified with over `HTTPS`. The node skips the route unless the bundle is only CERTIFICATE blocks that all parse.

    Maximum length: `65536`

  - **spec.registry.upstreams.upstream.mirrors.host**
    string

    Required value

    Registry host, optionally with a port.

    Length: `1..255`

  - **spec.registry.upstreams.upstream.mirrors.path**
    string

    Repository prefix inside the registry, for example `/virtualization`. The node prepends it to every repository it pulls through this address.

    Maximum length: `1024`

  - **spec.registry.upstreams.upstream.mirrors.scheme**
    string

    Protocol the node uses to connect to the registry.

Defaults to `HTTPS` when unset.

Allowed values: `HTTP`, `HTTPS`

- **spec.registry.upstreams.upstream.path**
  string

  Repository prefix inside the registry, for example `/virtualization`. The node prepends it to every repository it pulls through this address.

  Maximum length: `1024`

- **spec.registry.upstreams.upstream.scheme**
  string

  Protocol the node uses to connect to the registry.

  Defaults to `HTTPS` when unset.

  Allowed values: `HTTP`, `HTTPS`

- **spec.registryPackagesProxy**
  object

  How the node reaches `registry-packages-proxy`, which serves it extensions and operating system images.

  - **spec.registryPackagesProxy.accessToken**
    string

    Bearer token the node presents to the proxy.

    Base64-encoded in the object.

    Maximum length: `4096`

  - **spec.registryPackagesProxy.endpoints**
    array of strings

    `registry-packages-proxy` URLs with scheme and port.

    When unset, the node uses port `4219` of the `apiServerEndpoints` hosts. Set only in the initial configuration of the first master node.

    - **Element of the array**
      string

      Pattern: `^https?://(\[[0-9A-Fa-f:]+\]|[A-Za-z0-9]([-A-Za-z0-9]*[A-Za-z0-9])?([.][A-Za-z0-9]([-A-Za-z0-9]*[A-Za-z0-9])?)*):(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|6[0-4][0-9]{3}|[1-5][0-9]{4}|[1-9][0-9]{0,3})/?$`

      Maximum length: `253`

- **spec.staticPods**
  array of objects

  Manifests of static pods that kubelet runs without using the Kubernetes scheduler.

  The node writes the manifests to `/etc/kubernetes/manifests`. Static pods must obtain any other required resources on their own. For example, a pod can issue its own certificates, use the node's kubeconfig, and mount directories for persistent state using `hostPath` with the `DirectoryOrCreate` type.

  Control plane component manifests created by the node agent are not included in `spec.staticPods` and are not modified when this parameter is processed.

  Static pod.

  - **spec.staticPods.manifest**
    string

    Required value

    Complete pod manifest.

    When the manifest is written to disk, the `$MY_IP` substring is replaced with the node's IP address, similarly to control plane component manifests.

Length: `1..32768`

- **spec.staticPods.name**
  string

  Required value

  Manifest file name without an extension.

  Used as the list item key and is independent of `metadata.name` of the pod in the manifest. Kubelet determines the mirror pod name from the manifest rather than the file name.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Length: `1..63`

- **spec.staticPods.requestedBy**
  object

  Object that requested the static pod.

  Unset for a static pod of the platform itself.

  - **spec.staticPods.requestedBy.kind**
    string

    Required value

    Kind of the object.

    Allowed values: `NodeExtensionRequest`, `NodeStaticPodRequest`, `RegistryUpstream`, `ModuleSource`

  - **spec.staticPods.requestedBy.name**
    string

    Required value

    Name of the object.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Length: `1..253`

- **spec.statusToken**
  string

  StatusToken is the bearer presented to the node’s :50000 status port. Whoever writes the document mints it (dhctl, or the NodeConfigTemplate render for a hand-installed machine); the cluster carries it over untouched, having no way to tell the node a new one.

  Maximum length: `4096`

- **spec.storage**
  object

  Disk the operating system is installed on, and additional filesystems.

  The installation disk cannot be changed once set.

  - **spec.storage.device**
    string

    Device is the whole-disk block device to install onto, e.g. “/dev/sda”, “/dev/nvme0n1”, or a stable “/dev/disk/by-id/…” path. Ignored when diskSelector is set.

    Pattern: `^/dev/[A-Za-z0-9._/-]+$`

    Maximum length: `1024`

  - **spec.storage.diskSelector**
    object

    Selects the installation disk by attributes instead of a fixed path.

    All specified fields must match.

- **spec.storage.diskSelector.busPath**
  string

  BusPath matches the hardware bus path (glob).

  Maximum length: `253`

- **spec.storage.diskSelector.model**
  string

  Model matches the device model (glob), e.g. “Samsung*”.

  Maximum length: `253`

- **spec.storage.diskSelector.name**
  string

  Name matches the kernel device name (glob), e.g. “nvme0n1”.

  Maximum length: `253`

- **spec.storage.diskSelector.rotational**
  boolean

  Rotational matches spinning (true) vs solid-state (false) disks.

- **spec.storage.diskSelector.serial**
  string

  Serial matches the disk serial number (glob).

  Maximum length: `253`

- **spec.storage.diskSelector.size**
  string

  Size matches the disk capacity, optionally with a comparison operator, e.g. “>=100Gi”, “>1Ti”, “512Gi”.

  Maximum length: `253`

- **spec.storage.diskSelector.type**
  string

  Type matches the disk kind.

  Allowed values: `SSD`, `HDD`, `NVMe`, `SD`

- **spec.storage.diskSelector.wwid**
  string

  WWID matches the World Wide Identifier (glob).

  Maximum length: `253`

- **spec.storage.mounts**
  array of objects

  Mounts are additional filesystems the node makes available, at /mnt/ or wherever bindTo names. Nothing here is partitioned: only existing partitions and blank whole disks are formatted (when empty) and mounted.

  Mount is one additional filesystem, at /mnt/ unless bindTo says otherwise. Exactly one of device or partitionSelector names the partition; an empty one is formatted (label = name), an existing filesystem mounted as is.

  - **spec.storage.mounts.bindTo**
    string

    BindTo mounts the filesystem at this path instead of /mnt/, for a directory something else knows by name (/var/lib/etcd). The directory must be empty: mounting over files hides them, and hidden etcd data is lost data.

    Pattern: `^/[A-Za-z0-9._/-]+$`

    Maximum length: `1024`

  - **spec.storage.mounts.device**
    string

Device is the partition to use, e.g. “/dev/sdb1” or a stable “/dev/disk/by-id/…-part1” path.

Pattern: `^/dev/[A-Za-z0-9._/-]+$`

Maximum length: `1024`

- **spec.storage.mounts.filesystem**
  string

  Filesystem to create on an empty partition.

  Defaults to `Ext4` when unset.

  Allowed values: `Ext4`

- **spec.storage.mounts.mode**
  string

  Mode is the mode of the filesystem root after mounting, as an octal string, e.g. “0700”. Left alone when unset. A freshly made ext4 has its root at 0755, which is a mode etcd refuses to start on.

  Pattern: `^0[0-7]{3}$`

- **spec.storage.mounts.name**
  string

  Required value

  Name identifies the mount, and is both the mount point (/mnt/) and the filesystem label written when this node formats the partition. Capped at 16 characters because that is the size of the ext4 volume label field.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Length: `1..16`

- **spec.storage.mounts.partitionSelector**
  object

  PartitionSelector picks the partition by attributes instead of a fixed path.

  - **spec.storage.mounts.partitionSelector.blank**
    boolean

    Blank makes whole disks selectable, and only ones that carry nothing: no partition table, no filesystem — a cloud disk never touched. Without it a selector sees partitions only; a whole disk is where somebody’s layout lives.

  - **spec.storage.mounts.partitionSelector.fsType**
    string

    FSType matches the type of the filesystem already present (glob).

    Maximum length: `253`

  - **spec.storage.mounts.partitionSelector.label**
    string

    Label matches the filesystem label (glob).

    Maximum length: `253`

  - **spec.storage.mounts.partitionSelector.name**
    string

    Name matches the kernel device name (glob), e.g. “sdb1” or “nvme0n1p*”.

    Maximum length: `253`

  - **spec.storage.mounts.partitionSelector.partLabel**
    string

    PartLabel matches the GPT partition name (glob).

    Maximum length: `253`

  - **spec.storage.mounts.partitionSelector.partUUID**

string

PartUUID matches the GPT partition UUID exactly, ignoring case.

Maximum length: `253`

- **spec.storage.mounts.partitionSelector.size**
  string

  Size matches the size, optionally with a comparison operator, e.g. “>=100Gi”, “>1Ti”, “512Gi”. Without an operator the comparison is “>=”; “=” allows 1%, since a disk rarely reports an exact round size.

  Maximum length: `253`

- **spec.storage.mounts.partitionSelector.uuid**
  string

  UUID matches the filesystem UUID exactly, ignoring case.

  Maximum length: `253`

- **spec.storage.wipe**
  boolean

  Wipe controls whether an already-provisioned disk is wiped and re-partitioned. Default false: only an unprovisioned (or non-matching) disk is set up, so an existing layout stays and a reboot never destroys data.

- **spec.systemType**
  string

  Required value

  How the node is managed.

  - `DeckhouseEngine`: a Deckhouse Engine node. The node agent configures the whole node.
  - `GenericOS`: a node configured by bashible. The node agent manages only what `featureGates` turn on.

  The former names `Immutable` (for `DeckhouseEngine`) and `Mutable` (for `GenericOS`) are deprecated and accepted during the transition period.

  The node type cannot be changed. Replacing a former name with the new name of the same type, or back, is allowed.

  Allowed values: `Immutable`, `Mutable`, `DeckhouseEngine`, `GenericOS`

- **spec.updatePolicy**
  object

  UpdatePolicy controls how and when the node is updated.

  - **spec.updatePolicy.mode**
    string

    Update mode.

    The node checks the value and does not act on it: in either mode it updates within `windows`.

    Allowed values: `Manual`, `Automatic`

  - **spec.updatePolicy.windows**
    array of objects

    Time windows in UTC when the node may download and apply updates.

    An update is allowed if any window matches. Without windows, updates are allowed at any time.

    One update window, in UTC.

    - **spec.updatePolicy.windows.days**
      array of strings

      Days of the week the window applies to, as calendar days in UTC.

A time matches when its own day is listed, also in the part of a window that runs past midnight. When empty, the window applies every day.

- **Element of the array**
  string

  Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

- **spec.updatePolicy.windows.from**
  string

  Required value

  Start of the window, `H:MM` or `HH:MM` (24-hour, UTC).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

- **spec.updatePolicy.windows.to**
  string

  Required value

  End of the window, same format as `from`.

  A window whose end is before its start runs over midnight. The part after midnight belongs to its own calendar day: with `days: [Sat]`, a window from `22:00` to `04:00` allows Saturday 00:00 to 04:00 and Saturday 22:00 to midnight, and not Sunday 01:00.

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

- **status**
  object

  Status is reported by the on-node agent after each reconcile pass.

  - **status.agent**
    object

    Node agent build that wrote this status.

    - **status.agent.version**
      string

      Build version of the node agent.

  - **status.appliedGeneration**
    integer

    AppliedGeneration is the spec generation the node has applied.

    The node has applied the current spec when AppliedGeneration equals metadata.generation and ConfigurationApplied is True at that generation. The value stays behind while a disruptive change waits for approval, and while a new agent build installed for this generation has not applied it yet.

  - **status.conditions**
    array of objects

    Conditions of the node.

    A `False` condition whose reason ends with `Pending` is a wait. Any other `False` reason and `Unknown` mean a failure. `AddressResolved` does not affect `phase`.

    In `Ready`, `ConfigurationApplied` and `DisruptionApproved`, `observedGeneration` is the generation the node has processed. In the other conditions, it is the generation the node runs.

    Condition types:

    - `Ready`: `True` with reason `ReconcileSucceeded` when `phase` is `Ready`. Otherwise `False`, with the phase as the reason. The message names the condition that determined the phase.
    - `ConfigurationApplied`: `True` when the node runs the published configuration.

      With `True`, the reason is `ReconcileSucceeded`, or `UnknownFields` when the object has fields unknown to this agent build. The message lists them.

Reasons with `False` that mean a wait:

- `DisruptionApprovalPending`
- `Pending`
- `RestartPending`

Reasons with `False` that mean a failure:

- `Rejected`
- `RolledBackToLastKnownGood`
- `RollbackFailed`
- `PartiallyApplied`
- `FieldsNotApplicable`

- `DisruptionApproved`: Whether the interruption that the configuration of the node needs is approved.

  Reasons:

  - `ApprovalPending` (`False`): A configuration that interrupts the node waits for approval.
  - `AttemptsExhausted` (`False`): The interruptions `node-manager` attempted failed, and it makes no new attempt for now.
  - `Approved` (`True`): The interruption is approved.
  - `NoDisruptionPending` (`True`): Nothing waits for approval.

- `AddressResolved`: Whether the node has determined the address it registers with.

  Reasons with `False`:

  - `NoneInCIDR`, `Ambiguous`: The node cannot choose an address.
  - `AddressDrift`: The node keeps the address it registered with, but that address is no longer on the machine or the configuration now selects another one.

- Subsystem conditions, one per subsystem the pass ran or skipped:

  - `BootTimeSyncConfigured`
  - `APIEndpointsReachable`
  - `SysctlApplied`
  - `KernelModulesLoaded`
  - `MountsReady`
  - `ExtensionsReady`
  - `ContainerdReady`
  - `KubeletReady`
  - `StaticPodsReady`
  - `RegistryRoutesReady`
  - `UnitsRunning`
  - `OSImageUpToDate`
  - `ControlPlaneBootstrapped`

  Values of a subsystem condition:

  - `True`: The reason is `ReconcileSucceeded`, or a specific reason of the subsystem listed below.
  - `False`: The message gives the cause. The reason is `Pending` while the subsystem waits for something outside the node, otherwise `ReconcileFailed` or a more specific reason of the subsystem listed below.
  - `Unknown`: The subsystem did not run in this pass. The reason is `ReconcileSkipped`.

  Specific reasons with `True`:

  - `OwnedByRegistryAgent` (`RegistryRoutesReady`): The registry agent configures containerd’s registry hosts on this Generic OS node. The node writes no route and has removed the ones it wrote.

  Specific reasons with `False` that mean a failure:

  - `StaticPodFailed` (`StaticPodsReady`): A static pod manifest is not written or not removed. The `staticPods` entry gives the cause.
  - `OSImageRolledBack` (`OSImageUpToDate`): The node rolled back from the image or refused it before. The image is not tried again.
  - `OSImageRejected` (`OSImageUpToDate`): The kernel refused the root hash signature of the image. The image is not tried again.
  - `OSImageRootHashMismatch` (`OSImageUpToDate`): The image does not carry the root hash the configuration declares. The image is not tried again.

- `OSImageDownloadFailed` (`OSImageUpToDate`): The node could not download the image. The node retries.
- `OSImageStaged` (`OSImageUpToDate`): The image is staged for the next boot, but the node could not request the reboot.
- `OSImageOnTrial` (`OSImageUpToDate`): The node did not confirm the new image in time and could not request the reboot that tries it again.

Waits of `OSImageUpToDate` are published with the reason `Pending`, and the message tells them apart:

- `OSImageBaselineUnknown`: The node has no record of the image it runs and cannot update it until it is reinstalled.
- `OSImageUpdateDeferred`: The update waits for the update window.
- `OSImageStaged`, `OSImageOnTrial`: The node reboots into the new image or waits for its subsystems before it confirms the image.

Condition contains details for one aspect of the current state of this API Resource.

- **status.conditions.lastTransitionTime**
  string

  Required value

  lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**
  string

  Required value

  message is a human readable message indicating details about the transition. This may be an empty string.

  Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition’s last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-`

`9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

Maximum length: `316`

- **status.extensions**
  array of objects

  Extensions is the outcome of each configured system extension, one entry per extension. Republished on every pass, empty when the pass checked nothing — never “the previous outcome still holds”.

  ExtensionStatus is the reconcile outcome of one system extension.

  - **status.extensions.digest**
    string

    Digest depends on State.

    - Ready: the digest installed on the node.
    - Pending: the digest being installed. For the agent’s own entry, the digest installed and waiting for the new build to start.
    - Failed: the digest the node refused or could not install.
  - **status.extensions.message**
    string

    Message carries the cause when State is Failed.

  - **status.extensions.name**
    string

    Required value

    Name is the extension name: a spec.extensions[].name, or the name of an extension removed from spec whose uninstall failed on a Generic OS node.

  - **status.extensions.state**
    string

    Required value

    State of the extension.

    - Ready: the extension is installed (on a Deckhouse Engine node, also merged).
    - Pending: the extension is being fetched or waits for the update window, or the agent restarts with a new build of itself (reason RestartPending).
    - Failed: the cause is in Message.

    Allowed values: `Ready`, `Pending`, `Failed`

- **status.features**
  array of strings

  Features that ran on the node during its last reconciliation, sorted by name: the gates switched on in spec.featureGates plus the defaults of the node type.

  On a Deckhouse Engine node spec.featureGates is usually empty because unset gates are on there. This list shows what actually runs on the node.

- **status.lastReconcileTime**
  string

  LastReconcileTime is when the node last finished a reconcile pass — the only thing in the status that ages, so it is what tells a dead agent from a healthy one. Republished coarsely: judge staleness in tens of minutes.

- **status.maintenanceToken**
  string

  MaintenanceToken is the bearer token an operator presents to the node’s :50000 config-push port once the node has lost the API. Root-equivalent, so the apiserver keeps it out of every answer without the sensitive subresource, which nodes are not granted.

- **status.network**
  object

  Network is what the node resolved for itself: the interface it took its cluster address from, and that address. Written by the node alone.

- **status.network.address**
  string

  Address is the address kubelet registered the node with.

- **status.network.clusterInterface**
  string

  ClusterInterface is the NIC the address was taken from.

- **status.observedGeneration**
  integer

  ObservedGeneration is the latest spec generation the node has processed: it reaches the newest generation as soon as the node has looked at it, even while it is still held for approval (see AppliedGeneration).

- **status.osImage**
  object

  Status of the operating system image on the node. Filled in by the node.

  - **status.osImage.attemptsLeft**
    integer

    Number of remaining boot attempts with the image being verified before initramfs automatically rolls back to the previous image.

  - **status.osImage.digest**
    string

    Digest of the operating system image currently used by the node. The value is read from the node configuration partition.

    May be absent on nodes installed before the digest persistence mechanism was introduced.

  - **status.osImage.failedDigest**
    string

    Digest of an image that caused the node to roll back after booting. No further attempts are made to boot this image.

    Because a digest uniquely identifies immutable image content, a fixed version of the image has a different digest.

  - **status.osImage.rootHash**
    string

    Dm-verity root hash of the image currently used by the node. The value is read from the active A/B slot.

    Used to identify the actual root filesystem content independently of the image digest. For example, the digests of the desired and currently used images may differ after republishing, while identical `rootHash` values indicate that the root filesystem content has not changed.

  - **status.osImage.slot**
    string

    A/B slot (`a` or `b`) containing the image currently used by the node.

  - **status.osImage.trialDigest**
    string

    Digest of the image prepared for the next boot or being verified during the current boot.

    Set only during an update, while `digest` still contains the digest of the previous image.

- **status.phase**
  string

  Summary of the node.

  - `Ready`: The node runs the published configuration and every subsystem is healthy.
  - `Pending`: The node is healthy but does not run the published configuration yet.
  - `Degraded`: At least one condition other than `AddressResolved` reports a failure.

Allowed values: `Ready`, `Pending`, `Degraded`

- **status.staticPods**
  array of objects

  Results of processing static pods. A separate entry is created for each pod.

  Value is regenerated on each reconciliation and remains empty if no static pods were checked during the reconciliation.

  Processing result for a single static pod.

  - **status.staticPods.manifestDigest**
    string

    sha256 of the manifest as published in `spec`, in the form `sha256:<hex>`.

    Set when `state` is `Written`.

  - **status.staticPods.message**
    string

    Detailed information about the error when `state` is `Failed`.

  - **status.staticPods.name**
    string

    Required value

    Static pod name corresponding to `spec.staticPods[].name`.

  - **status.staticPods.reason**
    string

    Reason for the `Failed` state.

    Possible values:

    - `ManifestRejected`: Node rejected the manifest. The manifest in the resource must be corrected.
    - `WriteFailed`: Manifest file could not be written to the node. Check the node state.
    - `RemoveFailed`: Static pod was removed from `spec`, but the corresponding manifest file could not be removed from the node. The pod continues to run.

    Allowed values: `ManifestRejected`, `WriteFailed`, `RemoveFailed`

  - **status.staticPods.state**
    string

    Required value

    Manifest processing state.

    Possible values:

    - `Written`: File on disk matches the manifest from `spec`.
    - `Failed`: File does not match the manifest from `spec`, or the required operation could not be completed. The reason is specified in `reason`.

    Allowed values: `Written`, `Failed`

- **status.units**
  array of objects

  Units is the outcome of each managed systemd unit (containerd, kubelet and every unit an extension ships), one entry per unit. Republished on every pass like Extensions, and empty when the pass checked nothing.

  UnitStatus is the reconcile outcome of one managed systemd unit.

  - **status.units.message**
    string

    Message carries the cause when State is Failed.

  - **status.units.name**
    string

Required value

Name is the systemd unit name (e.g. containerd.service).

- **status.units.state**
  string

  Required value

  State is Active when the unit is running, Pending when it is queued to be started later this pass, or Failed with the cause in Message.

  Allowed values: `Active`, `Pending`, `Failed`

NodeExtensionRequest

Short names: ner

Scope: Cluster
Version: v1alpha1

Request to install an extension on the selected nodes: a system extension (a sysext image, optionally with kernel modules) or a registry package.

The requester sets the kind in `spec.extension.kind`. A registry package (`Package`) is installed on the selected nodes that bashible configures. A system extension (`SystemExtension`) is installed on the selected Deckhouse Engine nodes. The kind is never inferred from the image. If none of the selected groups installs extensions of this kind, the `Accepted` condition reports the `NotDeliverable` reason.

The image is addressed the same way as any package in the registry-packages-proxy: a name, a digest, and an optional repository and path. node-controller copies these fields unchanged into `spec.extensions` of the NodeConfig, and the node agent installs the extension. node-controller does not look up the image by module or ModuleSource.

Example:

```
# A system extension with kernel modules, fetched from the registry of a
ModuleSource.
apiVersion: deckhouse.io/v1alpha1
kind: NodeExtensionRequest
metadata:
  name: drbd
spec:
  extension:
    name: drbd
    digest: sha256:4f53cda18c2baa0c0354bb5f9a3ecbe5ed12ab4d8e11ba873c2f11161202b945
    kind: SystemExtension
    repository: registry.example.com/deckhouse/modules
    path: drbd/extensions
    kernelModules:
      - name: drbd
        params:
          - usermode_helper=disabled
      - name: drbd_transport_tcp
  nodeGroupSelector:
    matchNames:
      - worker
---
# A registry package from the cluster's main registry, on the nodes of every
# group bashible configures that carry the label.
apiVersion: deckhouse.io/v1alpha1
kind: NodeExtensionRequest
metadata:
  name: nvme-cli
spec:
  extension:
    name: nvme-cli
    digest: sha256:9b2a6f2bb7e3c5f6d1a0e4c8b7d6e5f4a3b2c1d0e9f8a7b6c5d4e3f2a1b0c9d8
    kind: Package
  nodeSelector:
    matchLabels:
      storage.example.com/nvme: "true"
```

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Required value

  - **spec.extension**
    object

    Required value

    Image to install and its kind. The image is addressed the same way as any package in the registry-packages-proxy.

    - **spec.extension.digest**
      string

      Required value

      Image manifest digest.

      Pattern: `^sha256:[a-f0-9]{64}$`

    - **spec.extension.kernelModules**
      array of objects

Kernel modules the node agent loads after merging the system extension, in the listed order. Allowed only with the `SystemExtension` kind. Each module is listed once.

Dependencies are not loaded automatically: list every module the extension needs, each one after the modules it depends on.

Adding a module interrupts each selected node: the node waits for an approved `Disruption` NodeOperation.

- **spec.extension.kernelModules.name**
  string

  Required value

  Kernel module name.

  Pattern: `^[a-zA-Z0-9][a-zA-Z0-9_-]*$`

  Length: `1..64`

- **spec.extension.kernelModules.params**
  array of strings

  Module parameters, passed as `key=value` strings.

  - **Element of the array**
    string

    Maximum length: `256`

- **spec.extension.kind**
  string

  Required value

  Image kind.

  - `Package`: a registry package. The node agent installs it on the selected nodes that bashible configures.
  - `SystemExtension`: a system extension. The node agent merges it on the selected Deckhouse Engine nodes.

  Nodes of groups of the other type do not receive the image. If none of the selected groups installs extensions of this kind, the `Accepted` condition reports the `NotDeliverable` reason.

  Cannot be changed after creation. To change the kind, delete the request and create it again.

  Allowed values: `Package`, `SystemExtension`

- **spec.extension.name**
  string

  Required value

  Extension name.

  The image of a system extension must contain the extension-release file for this name.

  The names `containerd`, `kubelet`, `kubernetes-cni`, `nodelet` and `qemu-guest-agent` belong to platform extensions and are not accepted.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Length: `1..63`

- **spec.extension.path**
  string

  The image's path within the repository (the proxy's `--rpp-path`). Left empty, the repository root is used.

  Maximum length: `512`

- **spec.extension.repository**
  string

  The key the registry-packages-proxy resolves credentials by (its `--rpp-repository`).

The key the registry packages proxy resolves credentials by (its `spec.repo`), for example a ModuleSource's `spec.registry.repo`. Left empty, the image is pulled from the cluster's main registry.

Pattern: `^[^[:space:]]+$`

Maximum length: `1024`

- **spec.nodeGroupSelector**
  object

  Narrows the extension to nodes of the named NodeGroups.

  - **spec.nodeGroupSelector.matchNames**
    array of strings

    The set of NodeGroup names the extension applies to.

    - **Element of the array**
      string

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

      Maximum length: `63`

- **spec.nodeSelector**
  object

  Narrows the extension to nodes carrying the given labels.

  - **spec.nodeSelector.matchLabels**
    object

    The set of node labels a node must carry to be selected.

- **status**
  object
  - **status.appliedNodes**
    integer

    Number of selected nodes that report the extension installed at the digest of this request.

    Allowed values: `0 <= X`

  - **status.conditions**
    array of objects

    Conditions of the request: `Accepted` and `Ready`.

    `Accepted` shows whether `node-manager` accepted the request. It is `True` with the `Accepted` reason or `False` with one of these reasons:

    - `InvalidExtension`: The extension has no name or no digest.
    - `ReservedName`: The name belongs to a platform extension.
    - `Conflict`: An older request of the same kind already uses this name or digest, or the platform installs a registry package with this name or digest on the same nodes.
    - `KernelModuleConflict`: An older request already loads one of the kernel modules with different parameters.
    - `NotDeliverable`: None of the selected NodeGroups installs extensions of this kind.

    `Ready` shows whether the selected nodes run the request's digest:

    - `True` with the `AllNodesApplied` reason: Every selected node runs it.
    - `True` with the `NoNodesSelected` reason: The request selects no node. The selected NodeGroups do not exist, have no nodes, or have no nodes that match `nodeSelector`.
    - `False` with the `RolloutInProgress` reason: Some selected nodes do not run it yet.
    - `False` with the `RefusedByNodes` reason: At least one node refused the extension.
    - `False` with the `NotAccepted` reason: The `Accepted` condition is `False`.
    - **status.conditions.lastTransitionTime**
      string

      Required value

    - **status.conditions.message**
      string

      Required value

      Maximum length: `32768`

        - **status.conditions.observedGeneration**
          integer

          Allowed values: `0 <= X`

        - **status.conditions.reason**
          string

          Required value

          Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

          Length: `1..1024`

        - **status.conditions.status**
          string

          Required value

          Allowed values: `True`, `False`, `Unknown`

        - **status.conditions.type**
          string

          Required value

          Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

          Maximum length: `316`

    - **status.failedNodes**
      integer

      Number of selected nodes that refused the extension at the request's digest.

      Allowed values: `0 <= X`

    - **status.matchedNodeGroups**
      array of strings

      The selected NodeGroups whose nodes install this kind of image: Deckhouse Engine groups for a system extension, groups bashible manages for a registry package.

    - **status.matchedNodes**
      integer

      Number of nodes in `matchedNodeGroups` that the selectors pick.

      Allowed values: `0 <= X`

    - **status.nodes**
      array of objects

      Selected nodes that have not applied the request, at most 32.

        - **status.nodes.message**
          string

          Why the node refused the request or what it waits for.

          Maximum length: `4096`

        - **status.nodes.name**
          string

          Required value

          Node name.

        - **status.nodes.state**
          string

          Required value

          `Failed` if the node refused the request or `Pending` if it has not answered yet.

          Allowed values: `Pending`, `Failed`

    - **status.observedGeneration**

integer

The generation of the spec this status reflects.

Allowed values: `0 <= X`

- **status.pendingNodes**
  integer

  Number of selected nodes that have reported neither the extension installed at the request's digest nor a refusal yet.

  Allowed values: `0 <= X`

- **status.phase**
  string

  Summary of the conditions.

  - `Pending`: The selected nodes are still installing the extension (the `Ready` condition has the `RolloutInProgress` reason).
  - `Ready`: The `Ready` condition is `True`.
  - `Degraded`: `node-manager` did not accept the request (the `Accepted` condition is `False`), or nodes refused the extension (the `Ready` condition has the `RefusedByNodes` reason).

  Allowed values: `Pending`, `Ready`, `Degraded`

NodeGroupConfiguration

Short names: ngc

Scope: Cluster
Version: v1alpha1

Executes bash scripts on nodes.

Read more in the module documentation.

- **spec**
  object

  Required value

  - **spec.bundles**
    array of strings

    Required value

    Bundles for step execution. You can set `'*'` for selecting all bundles.

    Examples:

    ```
    bundles:
    - ubuntu-lts
    - centos-7
    ```

    ```
    bundles:
    - ubuntu-lts
    ```

    ```
    bundles:
    - "*"
    ```

    - **Element of the array**
      string

      Allowed values: `ubuntu-lts`, `centos`, `debian`, `redos`, `rosa`, `astra`, `altlinux`, `opensuse`, `*`

  - **spec.content**
    string

    Required value

A bash script that does the same things you would do in a configuration step.

You can use Go Template to generate a script.

The list of parameters available for use in templates can be retrieved from the `bashible-apiserver-context` Secret as follows:

```
d8 k -n d8-cloud-instance-manager get secrets bashible-apiserver-context -o
jsonpath='{.data.input\.yaml}'| base64 -d
```

For example:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

You can also use the pre-defined bashbooster commands in the script. For example:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Example of using the proxy parameters of ClusterConfiguration resource:

```
bb-set-proxy
curl -I https://google.com
bb-unset-proxy
```

```
test-function-with-proxy() {
  bb-set-proxy
  trap bb-unset-proxy RETURN
  curl -I https://google.com
}
test-function-with-proxy
```

- **spec.nodeGroups**
  array of strings

  Required value

  List of NodeGroups to apply the step for. You can set `'*'` for selecting all NodeGroups.

  Examples:

  ```
  nodeGroups:
  - master
  - worker
  ```

  ```
  nodeGroups:
  - worker
  ```

  ```
  nodeGroups:
  - "*"
  ```

- **spec.weight**
  integer

  Order of the step execution.

  Default: `100`

NodeOperation

Short names: nop

Scope: Cluster
Version: v1alpha1

Describes an operation on a node that interrupts its work.

The `spec` of an existing operation cannot be changed, except `.spec.approved`. To approve a `Disruption` operation, run:

```
d8 k patch nodeoperation <OPERATION_NAME> --type=merge -p '{"spec":{"approved":true}}'
```

In the command, replace `<OPERATION_NAME>` with the name of the operation.

Operations for a deleted Cluster API Machine carry the label `node-manager.deckhouse.io/machine-uid`. Only `node-manager` can set it.

Example:

```
# Reboot a node.
apiVersion: deckhouse.io/v1alpha1
kind: NodeOperation
metadata:
  name: reboot-worker-1
spec:
  type: Reboot
  nodeName: worker-1
```

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Required value

  - **spec.approved**
    boolean

    Whether the interruption may go ahead. If the field is not set, the operation is not approved.

    Only meaningful for `Disruption`. `Reboot` and `Drain` ignore it.

    In the `Manual` approval mode of the NodeGroup (`spec.disruptions.approvalMode`), `node-manager` sets it to `false`, and an operator sets it to `true`. In the other modes, `node-manager` sets it to `true` on creation. An approval cannot be withdrawn.

  - **spec.cause**
    string

    Why the node is interrupted.

    For a `Disruption`, `node-manager` sets it on creation. For `Reboot` and `Drain`, an operator may set it.

    Maximum length: `1024`

  - **spec.configGeneration**
    integer

    Revision of the node's NodeConfig this disruption is for (`.metadata.generation` of the NodeConfig object).

    Allowed only for `Disruption`. The API server rejects the field for other types. If the NodeConfig moves to a newer generation before the operation is approved, `node-manager` deletes the operation.

    Allowed values: `1 <= X`

  - **spec.drain**

object

How to evict the workload before the node is interrupted.

Allowed for `Reboot` and `Disruption`. The API server rejects the field for `Drain`.

- **spec.drain.skip**
  boolean

  Interrupt the node without evicting its workload first.

  When `node-manager` creates a `Disruption`, it sets the field to `true` for these nodes:

  - The single node of the `master` NodeGroup
  - The node of the Deckhouse pod, while fewer than two nodes of its group are `Ready`
  - Every node of a NodeGroup with `spec.disruptions.automatic.drainBeforeApproval: false`

  Default: `false`

- **spec.nodeConfigUID**
  string

  NodeConfig object this disruption is for (`.metadata.uid` of the node's NodeConfig).

  Only meaningful for `Disruption`. Other types ignore it.

  Maximum length: `36`

- **spec.nodeName**
  string

  Required value

  The name of the node the operation applies to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Length: `1..253`

- **spec.type**
  string

  Required value

  Operation to perform on the node.

  - `Reboot`: Reboot the node. Only the nodes of a NodeGroup with `systemType: DeckhouseEngine` perform this operation. A node that bashible manages refuses it, and the operation fails with the reason `NodeRefused`.
  - `Drain`: Evict the workload from the node. The node stays unschedulable until an operator makes it schedulable again. `node-manager` also creates a `Drain` for the node of a deleted Cluster API Machine. `.spec.cause` names the machine.

    To cancel a running `Drain`, delete it. `node-manager` withdraws its eviction request from the node before the object is gone. The cordon the eviction placed stays: make the node schedulable again yourself.
  - `Disruption`: Interrupt the node's work so that the node can apply a new configuration.

  `node-manager` creates `Disruption` operations. Creating one by hand is refused. `Reboot` and `Drain` are meant for an operator.

  Allowed values: `Reboot`, `Drain`, `Disruption`

- **status**
  object
  - **status.bootID**
    string

    Boot ID of the node (`status.nodeInfo.bootID` of the Node object) when the reboot started.

    Maximum length: `64`

  - **status.conditions**
    array of objects

    Latest details of the operation's progress:

Latest details of the operation's progress:

- `Drained`: Written by `node-manager`. While the workload is being evicted, the condition is `False` with the reason `Draining`. Once the workload has left, it is `True` with the reason `Drained`. When the operation skips the eviction, it is `True` with the reason `DrainSkipped`.
- `EvictionBlocked`: Written by `node-manager` while evictions of the workload are refused, for example by a PodDisruptionBudget. The condition is `True` with the reason `EvictionRefused`, and its `message` lists the refusals as the eviction reports them, one per pod. The same text is in the operation's `message`. The condition is removed once the refusals stop or the eviction ends. A finished operation keeps the refusals it ended on.
- `NodeReported`: Written by the node. The condition is `True` with the reason `Succeeded`, `Released`, `Failed`, `Refused` or `RebootStarted`.
- `Retry`: Written by `node-manager` on a failed `Disruption`. The condition says whether `node-manager` makes another attempt for the same NodeConfig revision.

Reasons of the `Retry` condition:

- `Created` (`True`): The next attempt exists, and `message` names it.
- `Scheduled` (`True`): `node-manager` creates the next attempt at the time given in `message`.
- `AttemptsExhausted` (`False`): 3 attempts for the same revision have failed. `node-manager` makes no new attempt until the NodeConfig changes or the oldest failed attempt is deleted.
- `Superseded` (`False`): The NodeConfig has moved to another generation or has been recreated.
- `NotRequested` (`False`): The node no longer asks to be interrupted for this revision.
- **status.conditions.lastTransitionTime**
  string

  Required value
- **status.conditions.message**
  string

  Required value

  Maximum length: `32768`
- **status.conditions.observedGeneration**
  integer

  Allowed values: `0 <= X`
- **status.conditions.reason**
  string

  Required value

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`
- **status.conditions.status**
  string

  Required value

  Allowed values: `True`, `False`, `Unknown`
- **status.conditions.type**
  string

  Required value

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.deadline**
  string

  Time when the current stage of the operation times out and the operation fails.

  The node gets 30 minutes from `startedAt` to report the result or to come back after a reboot. The eviction gets the drain timeout of the NodeGroup (`nodeDrainTimeoutSecond`)

plus 30 minutes.

- **status.drainDeadline**
  string

  When the eviction this operation asked for runs out of the time it was given.

  On the `Drain` of a deleted machine, the operation fails with `PreparationTimedOut` at this moment, and the machine is deleted with the pods that did not leave. The field is not set for a `Static` or `CloudStatic` NodeGroup without `nodeDrainTimeoutSecond`, and such a `Drain` has no time limit.

  It is pinned at the moment the eviction is requested, from the NodeGroup's `nodeDrainTimeoutSecond` as it stood then, because that is the bound the draining controller itself started running to. Deriving it again later would let a change to the NodeGroup cut short a drain that is still running.

- **status.eviction**
  object

  Progress of the eviction this operation asked for, written by `node-manager` while the eviction runs.

  The field appears once the pods to evict are listed and is updated every few seconds. It is removed when the eviction ends. A failed operation keeps the progress it ended on.

  - **status.eviction.remainingPods**
    integer

    Required value

    How many of the pods the eviction listed are still on the node, including the terminating ones. The eviction leaves DaemonSet pods, static pods and finished pods on the node, and they are not counted.

    Allowed values: `0 <= X`

  - **status.eviction.terminatingPods**
    array of strings

    Pods that have been evicted and have not stopped yet, as `<namespace>/<name>`.

- **status.finishedAt**
  string

  When the operation reached a terminal phase. A finished operation is kept as the record of what was done to the node and is collected once it is a day old.

- **status.message**
  string

  Human-readable explanation of `reason`.

  Maximum length: `32768`

- **status.nodeWasUnschedulable**
  boolean

  Whether the node was already out of the scheduler when the operation reached it.

  Releasing the node at the end of the operation restores this value instead of making the node schedulable, so an operation does not quietly undo a cordon an operator put there by hand.

- **status.phase**
  string

  Current phase of the operation, set by `node-manager`.

  - `Pending`: The operation is queued or waits for approval.
  - `Approved`: The operation may go ahead. The workload is being evicted from the node before the node receives the operation.
  - `InProgress`: For `Drain`, the workload is being evicted. For the other types, the node has been prepared and can perform the operation.
  - `Completed`: The operation finished successfully.
  - `Failed`: The operation failed.

  Allowed values: `Pending`, `Approved`, `InProgress`, `Completed`, `Failed`

- **status.reason**
  string

  Reason for the current phase of the operation.

  - `Pending`: `Queued` (nothing has been done yet), `WaitingForApproval`.
  - `Approved`: `Draining` (the workload is being evicted from the node), `WaitingForOperation` (another eviction of the node is running, and `message` names who requested it).
  - `InProgress`: `Draining` and `WaitingForOperation` (for `Drain`), `WaitingForNode` (the node has received the operation and has not reported yet), `Rebooting` (the node is rebooting and has not come back yet).
  - `Completed`: `Succeeded`, `Released` (the node gave up on the NodeConfig revision the `Disruption` is for), `Drained` (for `Drain`).
  - `Failed`: `NodeFailed` and `NodeRefused` (reported by the node), `PreparationTimedOut` (the workload did not leave the node in time), `NodeTimedOut` (the node did not report back or did not come back after a reboot in time), `NodeNotFound`, `NodeGroupMissing`, `MachineDeleted` (the machine of the node is being deleted).

  Pattern: `^[A-Za-z][A-Za-z0-9]*$`

  Maximum length: `128`

- **status.startedAt**
  string

  When the node was handed the operation. The wait for the node is measured from here.

NodeStaticPodRequest

Short names: nspr

Scope: Cluster
Version: v1alpha1

Request to run a static pod on the selected nodes without using the Kubernetes scheduler.

`node-manager` adds the manifest to `spec.staticPods` of the NodeConfig of each selected node. The node writes the manifest to disk, and kubelet runs the pod from it. The request reports whether the nodes wrote the manifest. It does not report whether the pod runs.

A user with permission to create NodeStaticPodRequests can run arbitrary manifests with root privileges on all selected nodes.

Example:

```
# A node-local agent on the nodes of the worker NodeGroup.
apiVersion: deckhouse.io/v1alpha1
kind: NodeStaticPodRequest
metadata:
  name: node-local-agent
spec:
  nodeGroupSelector:
    matchNames:
      - worker
  manifest: |
    apiVersion: v1
    kind: Pod
    metadata:
      name: node-local-agent
      namespace: kube-system
    spec:
      hostNetwork: true
      priorityClassName: system-node-critical
      containers:
        - name: agent
          image: deckhouse.local/images:node-local-agent
          args:
            - --listen-address=$MY_IP:5002
          env:
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: state
              mountPath: /var/lib/node-local-agent
      volumes:
        - name: state
          hostPath:
            path: /var/lib/node-local-agent
            type: DirectoryOrCreate
```

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Required value

  - **spec.manifest**
    string

    Required value

    Complete pod manifest.

    Requirements for the manifest:

    - `metadata.name` is a DNS subdomain (RFC 1123), `metadata.namespace` is a DNS label (RFC 1123).
    - The namespace is `kube-system` or starts with `d8-`.
    - `spec.nodeName` is not set.
    - The pod does not reference API objects. Not allowed: `spec.serviceAccountName`, volumes of the `secret`, `configMap`, `persistentVolumeClaim` and `projected` types, `envFrom` and `env[].valueFrom` with `secretKeyRef` or `configMapKeyRef` in containers and init containers.

    A manifest that does not meet these requirements is rejected on creation and update.

When the manifest is written to disk, the `$MY_IP` substring is replaced with the node's IP address.

Maximum manifest size is 32 KiB. A node can run no more than 16 static pods defined by NodeStaticPodRequests.

Length: `1..32768`

- **spec.nodeGroupSelector**
  object

  Restricts the pod to nodes in the specified NodeGroups. If no groups are specified, all NodeGroups are selected.

  The node agent writes the manifest from the NodeConfig of each selected node. On nodes managed by bashible, a resource named `registry-agent` or describing the `kube-system/registry-agent` pod is not applied, because bashible writes that manifest itself. If the resource reaches no other node, the `Ready` condition reports the `ServedByBashible` reason.

  - **spec.nodeGroupSelector.matchNames**
    array of strings

    Names of NodeGroups on whose nodes the pod must run.

    - **Element of the array**
      string

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

      Maximum length: `63`

- **status**
  object
  - **status.appliedNodes**
    integer

    Number of selected nodes that reported the current manifest written.

    Allowed values: `0 <= X`

  - **status.conditions**
    array of objects

    Conditions of the request: `Accepted` and `Ready`.

    `Accepted` shows whether `node-manager` accepted the request. It is `True` with the `Accepted` reason or `False` with one of these reasons:

    - `InvalidName`: The request name is not a DNS label (RFC 1123) of at most 63 characters.
    - `ReservedName`: The request name, or the `metadata.namespace` and `metadata.name` of the pod, is already used by a static pod of the platform.
    - `InvalidManifest`: The manifest does not meet the requirements of `spec.manifest`.
    - `Conflict`: An older request already defines a pod with the same `metadata.namespace` and `metadata.name`.
    - `LimitExceeded`: At least one selected NodeGroup already has 16 static pods from older requests. Only the nodes of the other selected NodeGroups get the manifest.

    `Ready` shows whether the selected nodes wrote the current manifest:

    - `True` with the `AllNodesApplied` reason: Every selected node wrote it.
    - `True` with the `NoNodesSelected` reason: The selector matches no existing NodeGroup, or the selected NodeGroups have no nodes.
    - `True` with the `ServedByBashible` reason: The selected NodeGroups are managed by bashible, which writes the `registry-agent` manifest on their nodes itself.
    - `False` with the `RolloutInProgress` reason: Some selected nodes have not written the current manifest yet.
    - `False` with the `RefusedByNodes` reason: At least one node refused the manifest.
    - `False` with the `NotAccepted` reason: The `Accepted` condition is `False`.
    - **status.conditions.lastTransitionTime**
      string

      Required value

    - **status.conditions.message**
      string

Required value

Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.failedNodes**
  integer

  Number of selected nodes that refused the manifest.

  Allowed values: `0 <= X`

- **status.matchedNodeGroups**
  array of strings

  NodeGroups that match the current selector.

- **status.matchedNodes**
  integer

  Number of nodes in the selected NodeGroups to which the node agent writes the manifest. Nodes of NodeGroups on which bashible writes the manifest itself are not counted.

  Allowed values: `0 <= X`

- **status.nodes**
  array of objects

  Selected nodes that have not written the current manifest, at most 32.

  - **status.nodes.message**
    string

    Why the node refused the manifest or what it waits for.

    Maximum length: `4096`

  - **status.nodes.name**
    string

    Required value

    Node name.

  - **status.nodes.state**
    string

    Required value

`Failed` if the node refused the manifest or `Pending` if it has not answered yet.

Allowed values: `Pending`, `Failed`

- **status.observedGeneration**
  integer

  Generation of `spec` that corresponds to the current `status`.

  Allowed values: `0 <= X`

- **status.pendingNodes**
  integer

  Number of selected nodes that have not yet reported either the current manifest written or a failure.

  Allowed values: `0 <= X`

- **status.phase**
  string

  Summary of the conditions.

  - `Pending`: The selected nodes are still writing the manifest (the `Ready` condition has the `RolloutInProgress` reason).
  - `Ready`: The `Ready` condition is `True`.
  - `Degraded`: `node-manager` did not accept the request (the `Accepted` condition is `False`), or nodes refused the manifest (the `Ready` condition has the `RefusedByNodes` reason).

  Allowed values: `Pending`, `Ready`, `Degraded`

NodeUser

Scope: Cluster

- v1
- v1alpha1

Defines the linux users to create on all nodes.

The user's home directory is created in the `/home/deckhouse/` directory.

- **spec**
  object

  Required value

  - **spec.extraGroups**
    array of strings

    Node user additional system groups.

    Examples:

    ```
    extraGroups:
    - docker
    ```

    ```
    extraGroups:
    - docker
    - ftp
    ```

  - **spec.isSudoer**
    boolean

    Persistence of node user in sudo group.

    Default: `false`

    Example:

    ```
    isSudoer: true
    ```

- **spec.nodeGroups**
  array of strings

  List of NodeGroups to apply the user for.

  Default: `["*"]`

  Examples:

  ```
  nodeGroups:
  - master
  - worker
  ```

  ```
  nodeGroups:
  - worker
  ```

  ```
  nodeGroups:
  - "*"
  ```

- **spec.passwordHash**
  string

  Hashed user password.

  The format corresponds to the password hashes in `/etc/shadow`. You can get it using the following command: `openssl passwd -6`.

  If the caller is not allowed to read the `nodeusers/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Example:

  ```
  passwordHash:
  "$2a$10$F9ey7zW.sVliT224RFxpWeMsgzO.D9YRG54a8T36/K2MCiT41nzmC"
  ```

- **spec.sshPublicKey**
  Deprecated
  string

  Node user SSH public key.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKey: ssh-rsa AAABBB
  ```

- **spec.sshPublicKeys**
  array of strings

  Node user SSH public keys.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKeys:
  - ssh-rsa AAABBB
  - cert-authority,principals="name" ssh-rsa BBBCCC
  ```

- **spec.uid**
  number

  Required value

  Node user ID.

  We recommend using the values `>= 1100` to avoid conflicts with manually created users.

  This parameter does not change during the entire resource life.

Allowed values: `1001 <= X`

Example:

```
uid: 1100
```

- **status**
  object

  User creation status information

  - **status.errors**
    object

    List of user creation errors on each node

Deprecated resource. Support for the resource might be removed in a later release.

Defines the linux users to create on all nodes.

The user's home directory is created in the `/home/deckhouse/` directory.

- **spec**
  object

  Required value

  - **spec.extraGroups**
    array of strings

    Node user additional system groups.

    Examples:

    ```
    extraGroups:
    - docker
    ```

    ```
    extraGroups:
    - docker
    - ftp
    ```

  - **spec.isSudoer**
    boolean

    Persistence of node user in sudo group.

    Default: `false`

    Example:

    ```
    isSudoer: true
    ```

  - **spec.nodeGroups**
    array of strings

    List of NodeGroups to apply the user for.

    Default: `["*"]`

    Examples:

    ```
    nodeGroups:
    - master
    - worker
    ```

    ```
    nodeGroups:
    - worker
    ```

```
nodeGroups:
- "*"
```

- **spec.passwordHash**
  string

  Hashed user password.

  The format corresponds to the password hashes in `/etc/shadow`. You can get it using the following command: `openssl passwd -6`.

  If the caller is not allowed to read the `nodeusers/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Example:

  ```
  passwordHash:
  "$2a$10$F9ey7zW.sVliT224RFxpWeMsgzO.D9YRG54a8T36/K2MCiT41nzmC"
  ```

- **spec.sshPublicKey**
  Deprecated
  string

  Node user SSH public key.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKey: ssh-rsa AAABBB
  ```

- **spec.sshPublicKeys**
  array of strings

  Node user SSH public keys.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKeys:
  - ssh-rsa AAABBB
  - cert-authority,principals="name" ssh-rsa BBBCCC
  ```

- **spec.uid**
  number

  Required value

  Node user ID.

  We recommend using the values `>= 1100` to avoid conflicts with manually created users.

  This parameter does not change during the entire resource life.

  Allowed values: `1001 <= X`

  Example:

  ```
  uid: 1100
  ```

SSHCredentials

Scope: Cluster

- v1alpha2
- v1alpha1

Contains credentials required by Cluster API Provider Static (CAPS) to connect over SSH. CAPS connects to the server (virtual machine) defined in the StaticInstance custom resource to manage its state.

A reference to this resource is specified in the credentialsRef parameter of the `StaticInstance` resource.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  SSHCredentialsSpec defines the desired state of SSHCredentials.

  - **spec.privateSSHKey**
    string

    Private SSH key in PEM format encoded as base64 string.

    If the user does not have permission to view sensitive SSHCredentials data, the API returns instead of the actual value.

  - **spec.sshExtraArgs**
    string

    A list of additional arguments to pass to the openssh command.

    Examples:

    ```
    sshExtraArgs: "-vvv"
    ```

    ```
    sshExtraArgs: "-c chacha20-poly1305@openssh.com"
    ```

    ```
    sshExtraArgs: "-c aes256-gcm@openssh.com"
    ```

    ```
    sshExtraArgs: "-m umac-64-etm@openssh.com"
    ```

    ```
    sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
    ```

  - **spec.sshPort**
    integer

    A port to connect to the host via SSH.

    Default: `22`

    Allowed values: `1 <= X <= 65535`

  - **spec.sudoPasswordEncoded**
    string

    Base64 encoded sudo password for the user.

    If the user does not have permission to view sensitive SSHCredentials data, the API returns instead of the actual value.

  - **spec.user**
    string

    Required value

    A username to connect to the host via SSH.

Contains credentials required by Cluster API Provider Static (CAPS) to connect over SSH. CAPS connects to the server (virtual machine) defined in the StaticInstance custom resource to manage its state.

A reference to this resource is specified in the credentialsRef parameter of the `StaticInstance` resource.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  SSHCredentialsSpec defines the desired state of SSHCredentials.

  - **spec.privateSSHKey**
    string

    Private SSH key in PEM format encoded as base64 string.

    If the user does not have permission to view sensitive SSHCredentials data, the API returns instead of the actual value.

  - **spec.sshExtraArgs**
    string

    A list of additional arguments to pass to the openssh command.

    Examples:

    ```
    sshExtraArgs: "-vvv"
    ```

    ```
    sshExtraArgs: "-c chacha20-poly1305@openssh.com"
    ```

    ```
    sshExtraArgs: "-c aes256-gcm@openssh.com"
    ```

    ```
    sshExtraArgs: "-m umac-64-etm@openssh.com"
    ```

    ```
    sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
    ```

  - **spec.sshPort**
    integer

    A port to connect to the host via SSH.

    Default: `22`

    Allowed values: `1 <= X <= 65535`

  - **spec.sudoPassword**
    string

    A sudo password for the user.

    If the user does not have permission to view sensitive SSHCredentials data, the API returns instead of the actual value.

  - **spec.user**
    string

Required value

A username to connect to the host via SSH.

StaticInstance

Scope: Cluster

- v1alpha2
- v1alpha1

StaticInstance describes a machine for the Cluster API Provider Static.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  StaticInstanceSpec defines the desired state of StaticInstance.

  - **spec.address**
    string

    Required value

    The IP address of the host.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **spec.credentialsRef**
    object

    Required value

    The reference to the `SSHCredentials` object.

    - **spec.credentialsRef.apiVersion**
      string

      API version of the referent.

    - **spec.credentialsRef.fieldPath**
      string

      If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

    - **spec.credentialsRef.kind**
      string

      Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

    - **spec.credentialsRef.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-

with-objects/names/#names

- **spec.credentialsRef.namespace**
  string

  Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

- **spec.credentialsRef.resourceVersion**
  string

  Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

- **spec.credentialsRef.uid**
  string

  UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status**
  object

  StaticInstanceStatus defines the observed state of StaticInstance.

  - **status.conditions**
    array of objects

    Conditions defines current service state of the StaticInstance.

    Condition contains details for one aspect of the current state of this API Resource.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

Required value

status of the condition, one of True, False, Unknown.

Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.currentStatus**
  object
  - **status.currentStatus.lastUpdateTime**
    string
  - **status.currentStatus.phase**
    string

    Allowed values: `Error`, `Pending`, `Bootstrapping`, `Running`, `Cleaning`

- **status.machineRef**
  object

  The reference to the `StaticMachine` object.

  - **status.machineRef.apiVersion**
    string

    API version of the referent.

  - **status.machineRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.machineRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **status.machineRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **status.machineRef.namespace**
    string

    Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

  - **status.machineRef.resourceVersion**
    string

    Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

  - **status.machineRef.uid**
    string

    UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status.nodeRef**
  object

  The reference to the `Node` object.

  - **status.nodeRef.apiVersion**
    string

    API version of the referent.

  - **status.nodeRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.nodeRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **status.nodeRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **status.nodeRef.namespace**
    string

    Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

  - **status.nodeRef.resourceVersion**
    string

    Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

  - **status.nodeRef.uid**
    string

    UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

StaticInstance describes a machine for the Cluster API Provider Static.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  StaticInstanceSpec defines the desired state of StaticInstance.

  - **spec.address**

string

Required value

The IP address of the host.

Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **spec.credentialsRef**
  object

  Required value

  The reference to the `SSHCredentials` object.

  - **spec.credentialsRef.apiVersion**
    string

    API version of the referent.

  - **spec.credentialsRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **spec.credentialsRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **spec.credentialsRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.credentialsRef.namespace**
    string

    Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

  - **spec.credentialsRef.resourceVersion**
    string

    Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

  - **spec.credentialsRef.uid**
    string

    UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status**
  object

  StaticInstanceStatus defines the observed state of StaticInstance.

  - **status.conditions**
    array of objects

    Conditions defines current service state of the StaticInstance.

    Condition contains details for one aspect of the current state of this API Resource.

    - **status.conditions.lastTransitionTime**
      string

      Required value

lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**
  string

  Required value

  message is a human readable message indicating details about the transition. This may be an empty string.

  Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.currentStatus**
  object
  - **status.currentStatus.lastUpdateTime**
    string
  - **status.currentStatus.phase**
    string

    Allowed values: `Error`, `Pending`, `Bootstrapping`, `Running`, `Cleaning`

- **status.machineRef**
  object

  The reference to the `StaticMachine` object.

  - **status.machineRef.apiVersion**
    string

    API version of the referent.

- **status.machineRef.fieldPath**
  string

  If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **status.machineRef.kind**
  string

  Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **status.machineRef.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **status.machineRef.namespace**
  string

  Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

- **status.machineRef.resourceVersion**
  string

  Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

- **status.machineRef.uid**
  string

  UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status.nodeRef**
  object

  The reference to the `Node` object.

  - **status.nodeRef.apiVersion**
    string

    API version of the referent.

  - **status.nodeRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.nodeRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **status.nodeRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **status.nodeRef.namespace**
    string

Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

- **status.nodeRef.resourceVersion**
  string

  Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

- **status.nodeRef.uid**
  string

  UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing nodes: FAQ

> The module lifecycle stage: General Availability

How do I add master nodes to a cloud cluster?

For the procedure of converting a cluster with a single master node to a multi-master cluster, refer to the FAQ of the `control-plane-manager` module.

How do I reduce the number of master nodes in a cloud cluster?

For the procedure of converting a multi-master cluster to a cluster with a single master node, refer to the FAQ of the `control-plane-manager` module.

Static nodes

You can add a static node to the cluster manually (an example) or by using Cluster API Provider Static.

How do I add a static node to a cluster (Cluster API Provider Static)?

To add a static node to a cluster (bare metal server or virtual machine), follow these steps:

1. Prepare the required resources:
   - Allocate a server or virtual machine and ensure that the node has the necessary network connectivity with the cluster.
   - If necessary, install additional operating system packages and configure the mount points that will be used on the node.
2. Create a user with `sudo` privileges:
   - Add a new user (in this example, `caps`) with `sudo` privileges:

     ```
     useradd -m -s /bin/bash caps
     usermod -aG sudo caps
     ```

   - Allow the user to run `sudo` commands without having to enter a password. For this, add the following line to the `sudo` configuration on the server (you can either edit the `/etc/sudoers` file, or run the `sudo visudo` command, or use some other method):

     ```
     caps ALL=(ALL) NOPASSWD: ALL
     ```

3. Set `UsePAM` to `yes` in `/etc/ssh/sshd_config` on server and restart sshd service:

   ```
   sudo systemctl restart sshd
   ```

4. Generate a pair of SSH keys with an empty passphrase on the server:

   ```
   ssh-keygen -t ed25519 -f caps-id -C "" -N ""
   ```

   The public and private keys of the caps user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.

5. Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory on the server:

   ```
   mkdir -p /home/caps/.ssh
   cat caps-id.pub >> /home/caps/.ssh/authorized_keys
   chmod 700 /home/caps/.ssh
   chmod 600 /home/caps/.ssh/authorized_keys
   chown -R caps:caps /home/caps/
   ```

6. Create the SSHCredentials resource.
7. Create the StaticInstance resource.
8. Create the NodeGroup resource with the `Static` nodeType, specify the desired number of nodes in the group and, if necessary, the filter for `StaticInstance`.

An example of adding a static node.

How do I add a batch of static nodes to a cluster manually?

Use an existing one or create a new NodeGroup custom resource (example of the `NodeGroup` called `worker`). The nodeType parameter for static nodes in the NodeGroup must be `Static` or `CloudStatic`.

You can automate the bootstrap process with any automation platform you prefer. The following is an example for Ansible.

1. Pick up one of Kubernetes API Server endpoints. Note that this IP must be accessible from nodes that are being added to the cluster:

   ```
   d8 k -n default get ep kubernetes -o json | jq '.subsets[0].addresses[0].ip + ":" +
   (.subsets[0].ports[0].port | tostring)' -r
   ```

   Check the K8s version. If the version >= 1.25, create `node-group` token:

   ```
   d8 k create token node-group --namespace d8-cloud-instance-manager --duration 1h
   ```

   Save the token you got and add it to the `token:` field of the Ansible playbook in the next steps.

2. If the Kubernetes version is smaller than 1.25, get a Kubernetes API token for a special ServiceAccount that Deckhouse manages:

   ```
   d8 k -n d8-cloud-instance-manager get $(d8 k -n d8-cloud-instance-manager get secret -o name |
   grep node-group-token) \
     -o json | jq '.data.token' -r | base64 -d && echo ""
   ```

3. Create Ansible playbook with `vars` replaced with values from previous steps:

```
- hosts: all
  become: yes
  gather_facts: no
  vars:
    kube_apiserver: <KUBE_APISERVER>
    token: <TOKEN>
  tasks:
    - name: Check if node is already bootsrapped
      stat:
        path: /var/lib/bashible
      register: bootstrapped
    - name: Get bootstrap secret
      uri:
        url: "https://{{ kube_apiserver }}/api/v1/namespaces/d8-cloud-instance-
manager/secrets/manual-bootstrap-for-{{ node_group }}"
        return_content: yes
        method: GET
        status_code: 200
        body_format: json
        headers:
          Authorization: "Bearer {{ token }}"
        validate_certs: no
      register: bootstrap_secret
      when: bootstrapped.stat.exists == False
    - name: Run bootstrap.sh
      shell: "{{ bootstrap_secret.json.data['bootstrap.sh'] | b64decode }}"
      args:
        executable: /bin/bash
      ignore_errors: yes
      when: bootstrapped.stat.exists == False
    - name: wait
      wait_for_connection:
        delay: 30
      when: bootstrapped.stat.exists == False
```

4. Specify one more `node_group` variable. This variable must be the same as the name of `NodeGroup` to which node will belong. Variable can be passed in different ways, for example, by using an inventory file.:

```
[system]
system-0
system-1

[system:vars]
node_group=system

[worker]
worker-0
worker-1

[worker:vars]
node_group=worker
```

5. Run the playbook with the inventory file.

How do I clean up a static node manually?

> **i** This method is valid for both manually configured nodes (using the bootstrap script) and nodes configured using CAPS.

To decommission a node from the cluster and clean up the server (VM), run the following command on the node:

For all nodes except the control-plane

1. Delete a node from the Kubernetes cluster:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

2. Run the cleanup script on the node:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

3. After restarting the node run the script `bootstrap.sh`.

For control-plane nodes

1. Remove the labels `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master`, and `node.deckhouse.io/group` from the node:

```
d8 k label node <node> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master-
node.deckhouse.io/group-
```

2. Make sure the removed node with control-plane has disappeared from the list of etcd cluster members:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ member list -w table
```

3. Delete a node from the Kubernetes cluster:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

4. Run the cleanup script on the node:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

5. After restarting the node run the script `bootstrap.sh`.
6. Wait for the Deckhouse queues to be processed and ensure that the etcd cluster member has reappeared in the list:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ member list -w table
```

Can I delete a StaticInstance?

A StaticInstance that is in the `Pending` state can be deleted with no adverse effects.

To delete a StaticInstance in any state other than `Pending` (`Running`, `Cleaning`, `Bootstrapping`), you need to:

1. Add the label `"node.deckhouse.io/allow-bootstrap": "false"` to the StaticInstance.

   Example command for adding a label:

```
d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
```

2. Wait until the StaticInstance status becomes `Pending`.

   To check the status of StaticInstance, use the command:

   ```
   d8 k get staticinstances
   ```

3. Delete the StaticInstance.

   Example command for deleting StaticInstance:

   ```
   d8 k delete staticinstance d8cluster-worker
   ```

4. Decrease the `NodeGroup.spec.staticInstances.count` field by 1.

How do I change the IP address of a StaticInstance?

You cannot change the IP address in the `StaticInstance` resource. If an incorrect address is specified in `StaticInstance`, you have to delete the StaticInstance and create a new one.

How do I migrate a manually configured static node under CAPS control?

You need to clean up the node, then hand over the node under CAPS control.

Why are the SSH key and the sudo password of SSHCredentials shown as `<omitted>`?

The `privateSSHKey`, `sudoPassword` (`v1alpha1`), and `sudoPasswordEncoded` (`v1alpha2`) fields of the SSHCredentials resource contain sensitive data and are protected from viewing.

If a user has no permissions to read sensitive SSHCredentials data, the API returns `<omitted>` instead of the actual value. The other fields — `user`, `sshPort`, `sshExtraArgs` — and the resource metadata remain available to users who are allowed to read SSHCredentials.

Additionally, kube-apiserver protects this data as follows:

- removes the `kubectl.kubernetes.io/last-applied-configuration` annotation from API responses if it may contain a copy of sensitive values;
- replaces sensitive values with `"******"` in audit events, regardless of user permissions and audit level;
- encrypts the resource in etcd using the same mechanism as for Kubernetes Secrets, if the `apiserver.encryptionEnabled` parameter of the `control-plane-manager` module is enabled.

Unmasked values are available to the CAPS controller, which needs them to connect to the node over SSH, as well as to users with the `SuperAdmin` access level and members of the `kubeadm:cluster-admins` group.

The `d8:subsystem:cluster:viewer` and `d8:subsystem:cluster:manager` roles allow reading the SSHCredentials resource, but do not grant access to the SSH key or the sudo password.

To check whether a user can read sensitive SSHCredentials data, run:

```
d8 k auth can-i get sshcredentials/sensitive --as=<user>
```

The `<omitted>` value means the actual value is hidden from the current user. This behavior applies to both `v1alpha1` and `v1alpha2`, so using a different API version does not reveal the hidden data.

How do I change the NodeGroup of a static node?

Note that if a node is under CAPS control, you **cannot** change the `NodeGroup` membership of such a node. The only alternative is to delete StaticInstance and create a new one.

To switch an existing manually created static node to another `NodeGroup`, you need to change its group label:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

Applying the changes will take some time.

How to clean up a node for adding to the cluster?

This is only needed if you have to move a static node from one cluster to another. Be aware these operations remove local storage data. If you just need to change a NodeGroup, follow this instruction.

> ⚠ Evict resources from the node and remove the node from LINSTOR/DRBD using the instruction if the node you are cleaning up has LINSTOR/DRBD storage pools.

1. Delete the node from the Kubernetes cluster:

   ```
   d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
   d8 k delete node <node>
   ```

2. Run cleanup script on the node:

   ```
   bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
   ```

3. Run the `bootstrap.sh` script after reboot of the node.

How do I know if something went wrong?

If a node in a nodeGroup is not updated (the value of `UPTODATE` when executing the `d8 k get nodegroup` command is less than the value of `NODES`) or you assume some other problems that may be related to the `node-manager` module, then you need to look at the logs of the `bashible` service. The `bashible` service runs on each node managed by the `node-manager` module.

To view the logs of the `bashible` service on a specific node, run the following command:

```
journalctl -fu bashible
```

Example of output when the `bashible` service has performed all necessary actions:

```
May 25 04:39:16 kube-master-0 systemd[1]: Started Bashible service.
May 25 04:39:16 kube-master-0 bashible.sh[1976339]: Configuration is in sync,
nothing to do.
May 25 04:39:16 kube-master-0 systemd[1]: bashible.service: Succeeded.
```

How do I know what is running on a node while it is being created?

You can analyze `cloud-init` to find out what's happening on a node during the bootstrapping process:

1. Find the node that is currently bootstrapping:

   ```
   d8 k get instances | grep Pending
   ```

   An example:

   ```
   d8 k get instances | grep Pending
   dev-worker-2a6158ff-6764d-nrtbj   Pending   46s
   ```

2. Get information about connection parameters for viewing logs:

   ```
   d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
   ```

   An example:

```
d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

3. Run the command you got (`curl -N http://192.168.199.158:8000` according to the example above) to see `cloud-init` logs and determine the cause of the problem on the node.

The logs of the initial node configuration are located at `/var/log/cloud-init-output.log`.

How do I update kernel on nodes?

Debian-based distros

Create a `Node Group Configuration` resource by specifying the desired kernel version in the `desired_version` variable of the shell script (the resource's spec.content parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

CentOS-based distros

Create a `Node Group Configuration` resource by specifying the desired kernel version in the `desired_version` variable of the shell script (the resource's spec.content parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="3.10.0-1160.42.2.el7.x86_64"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-dnf-install "kernel-${desired_version}"
```

NodeGroup parameters and their result

The NodeGroup parameter	Disruption update	Node provisioning	Kubelet restart
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+

The NodeGroup parameter	Disruption update	Node provisioning	Kubelet restart
kubelet.topologyManager	+ (policy other than `None`, details)	-	+
kubernetesVersion	-	-	+
nodeTemplate	-	-	-
static	-	-	+
update.maxConcurrent	-	-	-

Refer to the description of the NodeGroup custom resource for more information about the parameters.

When the `InstanceClass` or `instancePrefix` parameters are modified, the process is similar to updating a Deployment in Kubernetes (changing `InstanceClass` results in creating a new MachineSet, and changing `instancePrefix` results in creating a new MachineDeployment). New instances will be created according to the value of the `cloudInstances.maxSurgePerZone` parameter. As the new instances are created, the old ones will be gradually removed, with a drain operation performed beforehand.

During the disruption update, an evict of the pods from the node is performed. If any pod fails to evict, the evict is repeated every 20 seconds until a global timeout of 5 minutes is reached. After that, the pods that failed to evict are removed.

How do I redeploy ephemeral machines in the cloud with a new configuration?

If the Deckhouse configuration is changed (both in the node-manager module and in any of the cloud providers), the VMs will not be redeployed. The redeployment is performed only in response to changing `InstanceClass` or `NodeGroup` objects.

To force the redeployment of all Machines, you need to add/modify the `manual-rollout-id` annotation to the `NodeGroup`: `d8 k annotate NodeGroup name_ng "manual-rollout-id=$(uuidgen)" --overwrite`.

How do I allocate nodes to specific loads?

⚠ You cannot use the `deckhouse.io` domain in `labels` and `taints` keys of the `NodeGroup`. It is reserved for Deckhouse components. Please, use the `dedicated` or `dedicated.client.com` keys.

There are two ways to solve this problem:

1. You can set labels to `NodeGroup`'s `spec.nodeTemplate.labels`, to use them in the `Pod`'s spec.nodeSelector or spec.affinity.nodeAffinity parameters. In this case, you select nodes that the scheduler will use for running the target application.
2. You cat set taints to `NodeGroup`'s `spec.nodeTemplate.taints` and then remove them via the `Pod`'s spec.tolerations parameter. In this case, you disallow running applications on these nodes unless those applications are explicitly allowed.

i Deckhouse tolerates the `dedicated` by default, so we recommend using the `dedicated` key with any `value` for taints on your dedicated nodes.

To use custom keys for `taints` (e.g., `dedicated.client.com`), you must add the key's value to the array `.spec.settings.modules.placement.customTolerationKeys` parameters. This way, deckhouse can deploy system components (e.g., `cni-flannel`) to these dedicated nodes.

How do I give pods dedicated CPUs and NUMA alignment?

Use the `spec.kubelet.topologyManager` parameter of the NodeGroup. Deckhouse configures the CPU, memory and topology managers of the kubelet and calculates NUMA-dependent values on each node of the group.

The result depends on `policy`:

policy	What pods get	kubelet settings
`None` (default)	Nothing, all pods share all CPUs	Not set, the kubelet defaults are used
`BestEffort`	Exclusive CPUs for containers of Guaranteed pods that request a whole number of CPUs, NUMA-bound memory. NUMA alignment of CPUs, memory and devices when possible	`cpuManagerPolicy: static`, `memoryManagerPolicy: Static`, `topologyManagerPolicy: best-effort`
`Restricted`	Exclusive whole physical cores. The pod gets CPUs, memory and devices from the smallest possible set of NUMA nodes, otherwise it is rejected	`cpuManagerPolicy: static` with `full-pcpus-only`, `memoryManagerPolicy: Static`, `topologyManagerPolicy: restricted`
`SingleNumaNode`	Exclusive whole physical cores. CPUs, memory and devices come from one NUMA node, otherwise the pod is rejected	`cpuManagerPolicy: static` with `full-pcpus-only`, `memoryManagerPolicy: Static`, `topologyManagerPolicy: single-numa-node`

The `scope` parameter sets what is aligned for every policy except `None`: `Pod` (default) aligns all containers of a pod together, `Container` aligns each container separately.

The `enabled` parameter is deprecated and has no effect: the managers are controlled by `policy` alone. To turn them off, set `policy: None`.

For every policy except `None`, Deckhouse also:

- splits `reservedMemory` between NUMA nodes in proportion to their memory without hugepages;
- sets `prefer-closest-numa-nodes` for `Restricted` if the node reports a full NUMA distance table and its NUMA node IDs are consecutive from 0;
- sets `max-allowable-numa-nodes` if the node has more than 8 NUMA nodes.

The policies `BestEffort`, `Restricted` and `SingleNumaNode` require `resourceReservation.mode` `Auto`, or `Static` with a positive `static.cpu`.

With every policy except `None`, the memory of a Guaranteed pod is bound to the NUMA nodes chosen at admission, `BestEffort` included. A pod whose memory cannot be placed is rejected with `UnexpectedAdmissionError`. A container can be killed by the OOM killer when its NUMA nodes run out of memory, even if other NUMA nodes have free memory.

Exclusive CPUs are taken from the shared pool. Burstable and BestEffort pods, including system components, run on the CPUs that remain. This matters for pods that use more CPU than they request: without exclusive CPUs they use idle time of all cores, with exclusive CPUs they share the remaining CPUs. With `Auto`, the shared pool can shrink to 1 CPU. If needed, use `Static` with a larger `static.cpu`.

The following NodeGroup gives Guaranteed pods on the `worker-dpdk` nodes exclusive whole cores on one NUMA node:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-dpdk
spec:
  nodeType: Static
  kubelet:
    topologyManager:
      policy: SingleNumaNode
    resourceReservation:
      mode: Auto
```

Only containers of Guaranteed pods with a whole number of CPUs get exclusive CPUs: requests are equal to limits for CPU and memory in every container, and the CPU value is an integer:

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
spec:
  containers:
  - name: app
    image: registry.example.com/dpdk-app:latest
    resources:
      requests:
        cpu: "4"
        memory: 8Gi
      limits:
        cpu: "4"
        memory: 8Gi
```

Rules of the Restricted and SingleNumaNode policies

- The number of CPUs of a container must be a multiple of the number of threads per core. With SMT (Hyper-Threading) enabled, an odd number of CPUs is rejected with `SMTAlignmentError`.
- With `SingleNumaNode`, the pod (with `scope: Pod`) or each container (with `scope: Container`) must fit into one NUMA node. Size pods by the smallest NUMA node in the group.
- A pod that cannot be aligned is rejected: with `TopologyAffinityError` if its CPUs do not fit, with `UnexpectedAdmissionError` if its memory does not fit.
- The scheduler does not know about NUMA nodes. It can place a pod on a node where the pod is then rejected. The controller of the pod (for example, a Deployment or a StatefulSet) creates a new pod, which can be rejected again.

Changing the settings

Changing `policy`, or `scope` with a policy other than `None`, restarts the kubelet on the nodes of the group.

The node requests a disruption if all of the following is true:

- the new kubelet configuration uses a policy other than `None`;
- the kubelet CPU or memory settings change, for example `policy`, `scope`, the CPU or memory reservation in `resourceReservation`, or the amount of memory or hugepages on the node. Other reservations, such as `ephemeralStorage`, do not cause a disruption;
- the node runs Guaranteed pods. DaemonSet and static pods are not counted, because a drain does not evict them.

The disruption is handled according to the `disruptions` settings of the group:

- with `approvalMode: Automatic` the node is drained and then gets the new configuration;
- with `approvalMode: Automatic` and `automatic.drainBeforeApproval: false` the node is not drained. After the kubelet restart, running pods that do not fit the new policy are stopped;
- with `approvalMode: Manual` the node is not drained automatically. Drain it, then run `d8 k annotate node <NODE NAME> update.node.deckhouse.io/disruption-approved=`;
- with `approvalMode: RollingUpdate` (cloud nodes only) the node is replaced with a new one.

Switching to `None` does not drain nodes: running containers keep their CPUs and the NUMA node of their memory until they are recreated. To spread them over all CPUs, restart the workloads, for example with `d8 k rollout restart`.

Limitations

- Nodes with more than 8 NUMA nodes are supported, but pod admission takes longer.
- Nodes in one group may have different NUMA layouts. Each node gets its own values, but the pod size limit is set by the smallest NUMA node.
- NodeGroupConfiguration scripts must not change reservations or manager policies in `/var/lib/kubelet/config.yaml`.
- Allocate hugepages before the kubelet is configured. Steps of NodeGroupConfiguration run in the order of `weight` together with the built-in steps, and the kubelet is configured at step 64, so use a `weight` below 64. The NUMA split of the reservation counts hugepages allocated at that moment. Hugepages allocated later change the split and can cause one more drain if the node runs Guaranteed pods.

How to allocate nodes to system components?

Frontend

For Ingress controllers, use the `NodeGroup` with the following configuration:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/frontend: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

System components

`NodeGroup` for components of Deckhouse subsystems will look as follows:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/system: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: system
```

How do I speed up node provisioning on the cloud when scaling applications horizontally?

The most efficient way is to have some extra nodes “ready”. In this case, you can run new application replicas on them almost instantaneously. The obvious disadvantage of this approach is the additional maintenance costs related to these nodes.

Here is how you should configure the target `NodeGroup`:

1. Specify the number of “ready” nodes (or a percentage of the maximum number of nodes in the group) using the `cloudInstances.standby` parameter.
2. If there are additional service components on nodes that are not handled by Deckhouse (e.g., the `filebeat` DaemonSet), you can specify the percentage of node resources they can consume via the `standbyHolder.overprovisioningRate` parameter.
3. This feature requires that at least one group node is already running in the cluster. In other words, there must be either a single replica of the application, or the `cloudInstances.minPerZone` parameter must be set to `1`.

An example:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

How do I disable machine-controller-manager/CAPI in the case of potentially cluster-damaging changes?

> ✖ Use this switch only if you know what you are doing and clearly understand the consequences.

Set the `mcmEmergencyBrake` parameter to true:

```
mcmEmergencyBrake: true
```

For disabling CAPI, set the `capiEmergencyBrake` parameter to true:

```
capiEmergencyBrake: true
```

How do I restore the master node if kubelet cannot load the control plane components?

Such a situation may occur if images of the control plane components on the master were deleted in a cluster that has a single master node (e.g., the directory `/var/lib/containerd` was deleted). In this case, kubelet cannot pull images of the control plane components when restarted since the master node lacks authorization parameters required for accessing `registry.deckhouse.io`.

Below is an instruction on how you can restore the master node.

containerd

Execute the following command to restore the master node in any cluster running under Deckhouse:

```
d8 k -n d8-system get secrets deckhouse-registry -o json |
jq -r '.data.".dockerconfigjson"' | base64 -d |
jq -r '.auths."registry.deckhouse.io".auth'
```

Copy the command's output and use it for setting the `AUTH` variable on the corrupted master.

Next, you need to pull images of `control plane` components to the corrupted master:

```
for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
  crictl pull --auth $AUTH $image
done
```

You need to restart `kubelet` after pulling the images.

How to change CRI for NodeGroup?

> ⚠ CRI can only be switched from `Containerd` to `NotManaged` and back (the cri.type parameter).

Set NodeGroup `cri.type` to `Containerd` or `NotManaged`.

NodeGroup YAML example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

Also, this operation can be done with patch:

- For `Containerd`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":
  {"type":"Containerd"}}}'
  ```

- For `NotManaged`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":
  {"type":"NotManaged"}}}'
  ```

⚠ While changing `cri.type` for NodeGroups, created using `dhctl`, you must change it in `dhctl config edit provider-cluster-configuration` and in `NodeGroup` object.

After setting up a new CRI for NodeGroup, the node-manager module drains nodes one by one and installs a new CRI on them. Node update is accompanied by downtime (disruption). Depending on the `disruption` setting for NodeGroup, the node-manager module either automatically allows node updates or requires manual confirmation.

Why might the CRI change not apply?

When attempting to switch the CRI, the changes may not take effect. The most common reason is the presence of special node labels: `node.deckhouse.io/containerd-v2-unsupported` and `node.deckhouse.io/containerd-config=custom`.

The `node.deckhouse.io/containerd-v2-unsupported` label is set to a node if at least one of the following conditions is true:

- Kernel version lower than 5.8
- Linux kernel version is in the range 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS)
- systemd version lower than 244
- cgroup v2 is disabled
- EROFS file system is unavailable.

The `node.deckhouse.io/containerd-config=custom` label is set if the node contains `.toml` files in the `/etc/containerd/conf.d/` (if CRI containerd v1 is used on the cluster nodes) or `/etc/containerd/conf2.d/` (if CRI containerd v2 is used on the cluster nodes) directories. The `d8-gpu-*.toml` files of the `gpu` module are not taken into account. In this case, you should remove such files (provided this will not have critical impact on running containers) and delete the corresponding NodeGroupConfiguration (NGC) resources through which they may have been added.

If the Deckhouse Platform virtualization is used, an additional reason why the CRI may fail to switch can be the `containerd-dvcr-config.sh` NGC. If the virtualization platform is already installed and running, this NGC can be removed.

If you cannot remove the NodeGroupConfiguration resource that modifies the containerd configuration and is incompatible with containerd v2, use the universal template:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
spec:
  bundles:
  - '*'
  content: |
    {{- if eq .cri "ContainerdV2" }}
  # <Script to modify the configuration for ContainerdV2>.
    {{- else }}
  # <Script to modify the configuration for ContainerdV1>.
    {{- end }}
  nodeGroups:
  - '*'
  weight: 31
```

Additionally, to switch the CRI you may need to remove the custom label `node.deckhouse.io/containerd-config=custom`. You can do this with the following command:

```
for node in $(d8 k get nodes -l node-role.kubernetes.io/<Name of NodeGroup where CRI is changed>=);
do kubectl label $node node.deckhouse.io/containerd-config-; done
```

How to change CRI for the whole cluster?

⚠ CRI can only be switched from `Containerd` to `NotManaged` and back (the cri.type parameter).

It is necessary to use the `dhctl` utility to edit the `defaultCRI` parameter in the `cluster-configuration` config.

Also, this operation can be done with the following patch:

- For `Containerd`:

```
data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r
'.data."cluster-configuration.yaml"' | base64 -d | sed "s/NotManaged/Containerd/"
| base64 -w0)"
d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-
configuration.yaml\":\"$data\"}}"
```

- For `NotManaged`:

```
data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r
'.data."cluster-configuration.yaml"' | base64 -d | sed "s/Containerd/NotManaged/"
| base64 -w0)"
d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-
configuration.yaml\":\"$data\"}}"
```

If it is necessary to leave some NodeGroup on another CRI, then before changing the `defaultCRI` it is necessary to set CRI for this NodeGroup, as described here.

✖ Changing `defaultCRI` entails changing CRI on all nodes, including master nodes. If there is only one master node, this operation is dangerous and can lead to a complete failure of the cluster! The preferred option is to make a multi-master and change the CRI type.

When changing the CRI in the cluster, additional steps are required for the master nodes:

1. Deckhouse updates nodes in master NodeGroup one by one, so you need to discover which node is updating right now:

```
d8 k get nodes -l node-role.kubernetes.io/control-plane="" -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") |
.metadata.name' -r
```

2. Confirm the disruption of the master node that was discovered in the previous step:

```
d8 k annotate node <master node name> update.node.deckhouse.io/disruption-approved=
```

3. Wait for the updated master node to switch to `Ready` state. Repeat steps for the next master node.

How to add node configuration step?

Additional node configuration steps are set via the NodeGroupConfiguration custom resource.

How to automatically put custom labels on the node?

1. On the node, create the directory `/var/lib/node_labels`.
2. Create a file or files containing the necessary labels in it. The number of files can be any, as well as the number of subdirectories containing them.
3. Add the necessary labels to the files in the `key=value` format. For example:

```
example-label=test
```

4. Save the files.

When adding a node to the cluster, the labels specified in the files will be automatically affixed to the node.

⚠ Please note that it is not possible to add labels used in Deckhouse Platform in this way. This method will only work with custom labels that do not overlap with those reserved for Deckhouse.

How to deploy custom containerd configuration?

ℹ The example of `NodeGroupConfiguration` uses functions of the script 032_configure_containerd.sh.

✖ Adding custom settings causes a restart of the containerd service.

Bashible on nodes merges main Deckhouse Platform containerd configuration with the following configuration files:

- `/etc/containerd/conf.d/*.toml`: If containerd v1 is used as the CRI on the cluster nodes.
- `/etc/containerd/conf2.d/*.toml`: If containerd v2 is used as the CRI on the cluster nodes.

⚠ You can override the values of the parameters that are specified in the file `/etc/containerd/deckhouse.toml`, but you will have to ensure their functionality on your own. Also, it is better not to change the configuration for the master nodes (nodeGroup `master`).

The following are configuration examples of the NodeGroupConfiguration resources adding a custom configuration file for a corresponding containerd version.

- For containerd v1
- For containerd v2

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
      [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-
shm.options]
        BinaryName = ""
        SystemdCgroup = true
        ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

How to add configuration for an additional registry?

Containerd supports two methods for registry configuration: the **deprecated** method and the **actual** method.

To check for the presence of the **deprecated** configuration method, run the following commands on the cluster nodes:

```
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.configs'

# Example output:
# [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
#   [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
# [plugins."io.containerd.grpc.v1.cri".registry.configs]
#   [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]
```

To check for the presence of the **actual** configuration method, run the following command on the cluster nodes:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'

# Example output:
# config_path = "/etc/containerd/registry.d"
```

Old Method

⚠ This containerd configuration format is deprecated.

i Used in containerd v1 when Deckhouse is not managed by the Registry module.

The configuration is described in the main containerd configuration file `/etc/containerd/config.toml`.

Adding custom configuration is carried out through the `toml merge` mechanism. Configuration files from the `/etc/containerd/conf.d` directory are merged with the main file `/etc/containerd/config.toml`. The merge takes place during the execution of the `032_configure_containerd.sh` script, so the corresponding files must be added in advance.

Example configuration file for the `/etc/containerd/conf.d/` directory:

```
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
          endpoint = ["https://${REGISTRY_URL}"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
          auth = "${BASE_64_AUTH}"
          username = "${USERNAME}"
          password = "${PASSWORD}"
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
          ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
          insecure_skip_verify = true
```

✖ Adding custom settings through the `toml merge` mechanism causes the containerd service to restart.

How to add additional registry auth (deprecated method)?

Example of adding authorization to a additional registry when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

How to configure a certificate for an additional registry (deprecated method)?

Example of configuring a certificate for an additional registry when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"


    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
            ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
```

> **i** In addition to containerd, the certificate can be added into the OS.

How to add TLS skip verify (deprecated method)?

Example of adding TLS skip verify when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
            insecure_skip_verify = true
    EOF
```

After applying the configuration file, verify access to the registry from the nodes using the command:

```
# Via the CRI interface
crictl pull private.registry.example/image/repo:tag
```

How to set up a mirror for public image registries (deprecated method)?

Example of configuring a mirror for public image registries when using the **deprecated** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
              endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
              endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]
    EOF
```

New Method

> **i** Used in containerd v2.
>
> Used in containerd v1 when managed through the `registry` module (for example, in `Direct` mode).

The configuration is defined in the `/etc/containerd/registry.d` directory.
Configuration is specified by creating subdirectories named after the registry address:

```
/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.io
    ├── ca.crt
    └── hosts.toml
```

Example contents of the `hosts.toml` file:

```
[host]
  # Mirror 1.
  [host."https://${REGISTRY_URL_1}"]
    capabilities = ["pull", "resolve"]
    ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

    [host."https://${REGISTRY_URL_1}".auth]
      username = "${USERNAME}"
      password = "${PASSWORD}"

  # Mirror 2.
  [host."http://${REGISTRY_URL_2}"]
    capabilities = ["pull", "resolve"]
    skip_verify = true
```

> **i** Configuration changes do not cause the containerd service to restart.

How to add additional registry auth (actual method)?

Example of adding authorization to a additional registry when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        [host."https://${REGISTRY_URL}".auth]
          username = "username"
          password = "password"
    EOF
```

How to configure a certificate for an additional registry (actual method)?

Example of configuring a certificate for an additional registry when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        ca = ["/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt"]
    EOF
```

i In addition to containerd, the certificate can be added into the OS.

How to add TLS skip verify (actual method)?

Example of adding TLS skip verify when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not required.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        skip_verify = true
    EOF
```

After applying the configuration file, check access to the registry from the nodes using the following commands:

```
# Via the CRI interface.
crictl pull private.registry.example/image/repo:tag

# Via ctr with the configuration directory specified.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/image/repo:tag

# Via ctr for an HTTP registry.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/image/repo:tag
```

How to set up a mirror for public image registries (actual method)?

Example of configuring a mirror for public image registries when using the **actual** configuration method:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
```

How to use NodeGroup’s priority feature

The priority field of the `NodeGroup` CustomResource allows you to define the order in which nodes will be provisioned in the cluster. For example, `cluster-autoscaler` can first provision *spot-nodes* and switch to regular ones when they run out. Or it can provision larger nodes when there are plenty of resources in the cluster and then switch to smaller nodes once cluster resources run out.

Here is an example of creating two `NodeGroups` using spot-node nodes:

```
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral
```

In the above example, `cluster-autoscaler` will first try to provision a spot-node. If it fails to add such a node to the cluster within 15 minutes, the `worker-spot` NodeGroup will be paused (for 20 minutes), and `cluster-autoscaler` will start provisioning nodes from the `worker` NodeGroup. If, after 30 minutes, another node needs to be deployed in the cluster, `cluster-autoscaler` will first attempt to provision a node from the `worker-spot` NodeGroup before provisioning one from the `worker` NodeGroup.

Once the `worker-spot` NodeGroup reaches its maximum (5 nodes in the example above), the nodes will be provisioned from the `worker` NodeGroup.

Note that node templates (labels/taints) for `worker` and `worker-spot` NodeGroups must be the same (or at least suitable for the load that triggers the cluster scaling process).

How to interpret Node Group states?

Ready — the node group contains the minimum required number of scheduled nodes with the status `Ready` for all zones.

Example 1. A group of nodes in the `Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
  - status: "True"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Example 2. A group of nodes in the `Not Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
  - status: "False"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Updating — a node group contains at least one node in which there is an annotation with the prefix `update.node.deckhouse.io` (for example, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval - a node group contains at least one node that has an annotation `update.node.deckhouse.io/disruption-required` and there is no annotation `update.node.deckhouse.io/disruption-approved`.

Scaling — calculated only for node groups with the type `CloudEphemeral`. The state `True` can be in two cases:

1. When the number of nodes is less than the *desired number of nodes* in the group, i.e. when it is

necessary to increase the number of nodes in the group.
2. When a node is marked for deletion or the number of nodes is greater than the *desired number of nodes*, i.e. when it is necessary to reduce the number of nodes in the group.

The *desired number of nodes* is the sum of all replicas in the node group.

Example. The desired number of nodes is 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
...
  desired: 2
...
```

Error — contains the last error that occurred when creating a node in a node group.

How do I make werf ignore the Ready conditions in a node group?

werf checks the `Ready` status of resources and, if available, waits for the value to become `True`.

Creating (updating) a nodeGroup resource in a cluster can take a significant amount of time to create the required number of nodes. When deploying such a resource in a cluster using werf (e.g., as part of a CI/CD process), deployment may terminate when resource readiness timeout is exceeded. To make werf ignore the nodeGroup status, the following `nodeGroup` annotations must be added:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

What is an Instance resource?

An Instance resource contains a description of an implementation-independent ephemeral machine resource. For example, machines created by MachineControllerManager or Cluster API Provider Static will have a corresponding Instance resource.

The object does not contain a specification. The status contains:

1. A link to the InstanceClass if it exists for this implementation;
2. A link to the Kubernetes Node object;
3. Current machine status;
4. Information on how to view machine creation logs (at the machine creation stage).

When a machine is created/deleted, the Instance object is created/deleted accordingly. You cannot create an Instance resource yourself, but you can delete it. In this case, the machine will be removed from the cluster (the removal process depends on implementation details).

When is a node reboot required?

Node reboots may be required after configuration changes. For example, after changing certain sysctl settings, specifically when modifying the `kernel.yama.ptrace_scope` parameter (e.g., using `astra-ptrace-lock enable/disable` in the Astra Linux distribution).

How to enable a delay before a node shutdown or restart while critical pods are running on it?

> ⚠ To decide whether to block a node shutdown, Deckhouse Platform additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

To enable the mechanism that delays a pod's restart or shutdown, add the label `pod.deckhouse.io/inhibit-node-shutdown` to the Pod (for a Deployment, specify the label in the pod template).

Example of a Pod manifest with the `pod.deckhouse.io/inhibit-node-shutdown` label:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```

> ⚠ The block only applies to pods labeled with `pod.deckhouse.io/inhibit-node-shutdown` that are in the `Running` phase on the node. Kubelet shuts down the pods without this label according to the standard graceful shutdown process.

Searching pods that prevent shutdown and performing the node troubleshooting

To search the pods that are preventing a node shutdown, use the following command:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -o wide
```

To view the `GracefulShutdownPostpone` condition on a node, use the following command:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?
(@.type=="GracefulShutdownPostpone")]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}
{.message}{"\n"}{end}'
```

Possible values of the `reason` field:

- `WaitingForShutdownSignal`: The shutdown blocking mechanism is running on the node, but the blocking has not yet begun.
- `PodsWithLabelAreRunningOnNode`: The shutdown has been blocked because there are pods with the label `pod.deckhouse.io/inhibit-node-shutdown` still running on the node.
- `NoRunningPodsWithLabel`: The blocking has been released and the shutdown can continue.

How do I set up fencing for a node group?

Fencing protects the cluster from nodes that lost the connection to it. A node that cannot reach the cluster stops its own work with a kernel panic. After that, Kubernetes can safely move the workloads to other nodes.

Set the fencing mode in the `spec.fencing.mode` parameter of the NodeGroup resource:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  fencing:
    mode: Normal
```

Each mode selects a built-in FencingSLAProfile. The profile sets the timings of the failure detection and the watchdog timeout:

Mode	Profile	Time to consider the control plane lost (`controlPlaneLostAfter`)	Watchdog timeout	Reaction to a failure
`Disabled`	none	none	none	Fencing is off. This is the default mode.
`Critical`	`critical`	4.15 s	5 s	About 12 s
`Normal`	`normal`	9.1 s	15 s	About 30 s
`Slow`	`slow`	20.2 s	30 s	About 60 s

The reaction time is a target. The tolerance is ±20 %. A failed node stops not later than `controlPlaneLostAfter` + 0.2 s + the watchdog timeout after the failure: 9.35 s with `Critical`, 24.3 s with `Normal` and 50.4 s with `Slow`.

A short timeout gives a fast reaction. But a short timeout also gives a higher risk of a false panic when the cluster is slow. Use `Slow` for groups with an unstable network or a heavy load. Use `Critical` only when a fast reaction is more important than the risk of a false panic.

Note the following:

- The `Watchdog` value of the `spec.fencing.mode` parameter is deprecated. It works as `Slow`. Use `Normal`, `Slow`, or `Critical` (`Critical` is available only in the commercial editions).
- The `spec.fencing.watchdog.timeout` parameter is deprecated. Deckhouse ignores it. The watchdog timeout comes from the profile of the mode.
- In a cluster with one master node, Deckhouse does not let you turn on fencing. Refer to Fencing in a cluster with one master node.
- Deckhouse does not let you change the mode of a node group while a node of the group has a FencingFailedNodeState object. Refer to Known limitations of the evacuation.
- The fencing agent runs on all nodes of the cluster when at least one node group has a mode other than `Disabled`. On the nodes of a `Disabled` group, the agent only takes part in the failure detection. It does not arm the watchdog.
- Fencing sets the `kernel.panic` kernel parameter to `0` on the nodes of the groups with a mode other than `Disabled`. A node does not restart by itself after a fencing panic.
- When you set a mode other than `Disabled` for a group, the new agent pod on a node starts only after bashible loads `softdog` and `/dev/watchdog` appears. Until then, the pod stays in the `ContainerCreating` state with the `FailedMount` event, and the agent does not work on the node. If the group already has agent pods, the DaemonSet replaces them by 10 percent of the nodes at a time. On Deckhouse Engine nodes, `softdog` is built into the kernel, and the pod does not wait. If a node waits for more than 5 minutes, the `D8FencingAgentDown` alert rises. When bashible loads `softdog`, the pod starts and the alert stops. No action is necessary. If the pod does not start, check the bashible logs on the node.

How do I check the fencing state of the nodes?

Run the following command:

```
d8 k get nodes -o custom-columns='NAME:.metadata.name,READY:.status.conditions[?
(@.type=="Ready")].status,FENCING:.status.conditions[?
(@.type=="Fencing")].reason,PROFILE:.metadata.annotations.node-
manager\.deckhouse\.io/fencing-profile'
```

The `READY` column shows the status of the `Ready` condition of the node. The `FENCING` column shows the reason of the `Fencing` condition, for example, `Armed` or `GroupDisabled`. The `PROFILE` column shows the `node-manager.deckhouse.io/fencing-profile` annotation of the node. The annotation has the FencingSLAProfile of the node group mode (`slow` for the deprecated `Watchdog` mode). This is also true when the watchdog of the node is not armed. If the group has the `Disabled` mode, the annotation is `disabled`. If the fencing agent does not run on the node, the `FENCING` and `PROFILE` columns show `<none>`. For example, this occurs when all node groups have the `Disabled` mode.

When fencing stops a node after the loss of the connection to the API server, the node cannot update its `Fencing` condition. Until the node restarts and the agent starts again, the `FENCING` column can still show `Armed`. To find a stopped node, look at the `READY` column: its value is not `True`.

What happens when the fencing agent restarts?

- A planned stop of the agent, for example, an update of the DaemonSet, a deletion of the pod or a shutdown of the node, disarms the watchdog. The node does not stop. If the agent has already stopped the reset of the timer after the loss of the control plane, the watchdog stays armed.
- If the agent crashes or is killed, the watchdog stays armed. The watchdog timeout starts when the agent process stops. At the start, the new agent process checks whether the control plane is available:
  - If the control plane is available, the agent disarms the watchdog. Then the agent arms the watchdog again, as at a usual start. Until then, the reason of the `Fencing` condition is not `Armed`.
  - If the control plane is not available, the agent does not disarm the watchdog. If the agent does not arm the watchdog again before the timeout expires, the node stops.
- If the agent crashes at each start and the control plane is available, the node usually continues to work, but its watchdog is not armed. The `D8FencingAgentRestarted` alert rises, and the reason of the `Fencing` condition is not `Armed`. Read the logs of the agent and remove the cause of the crashes.

What are the limits of softdog?

Fencing uses the `softdog` kernel module as the watchdog. Deckhouse loads the module with the `soft_panic=1` parameter. If the fencing agent does not reset the timer, the timer expires and the kernel panics.

`softdog` is a timer of the same kernel that it must protect. If the kernel has a hard lockup, the timer does not work and the node does not panic. If the node has a hardware watchdog, use it instead of `softdog`. A hardware watchdog works also when the kernel hangs.

Deckhouse Engine nodes

On nodes of groups with `systemType: Immutable` (Deckhouse Engine), `softdog` is built into the kernel. The NodeGroupConfiguration script does not run on these nodes.

If you installed a node before the Deckhouse Engine release with the built-in `softdog`, the node does not have the built-in `softdog`. The node gets `softdog` only after you reinstall it or update its boot partition.

How does fencing work on master nodes?

The fencing agent treats master nodes in the same way as worker nodes. The same modes and conditions apply. There is no special rule for the `master` group.

Note the following:

- After a fencing panic, a master node does not start again by itself. Return the node to the cluster manually. If etcd runs on the node, return the node to the etcd cluster manually.
- When a master node fails, the taint comes later than on a worker node, usually by 10–20 seconds. kube-apiserver on the other master nodes also sends requests to etcd on the failed master node and waits for an answer for approximately 10 seconds. The requests of the fencing-controller wait too. No action is necessary.
- The fencing-controller runs on the master nodes. If the master node of the controller fails or is not ready, Kubernetes evicts the controller pod from it at once, and the controller starts on another master node. In this case, the taint of a failed node comes later. The delay is the time in which Kubernetes marks the master node as unreachable (`nodeMonitorGracePeriodSeconds` of the `control-plane-manager` module, 40 seconds by default), plus the time to start the pod. In a cluster without the high availability mode, the controller has one replica.

Fencing in a cluster with one master node

A node with fencing stops when it loses the control plane. In a cluster with one master node, a failure or a restart of this master node stops all nodes with fencing. Also, when the master node stops, no component remains to set the taint and to move the workloads. Thus, Deckhouse does not let you turn on fencing in a cluster with one master node:

- Deckhouse refuses a change of a NodeGroup that turns fencing on, that is, a change of `spec.fencing.mode` from `Disabled` to another mode.
- When you create a NodeGroup with a mode other than `Disabled`, Deckhouse only shows a warning. This does not stop the installation of a cluster with fencing in its resources.
- A NodeGroup that already has fencing on stays editable. Deckhouse does not turn off fencing that is already on.

Deckhouse counts the master nodes as follows. It counts the nodes with the `node-role.kubernetes.io/control-plane` label. If there are no such nodes, it counts the system nodes with the `node-role.deckhouse.io/master` label.

Until the installation of the cluster is complete, Deckhouse only shows a warning also when you turn fencing on. For this check, the installation is complete when the first node that is not a master node becomes `Ready`. Thus, in a cluster of one master node without other nodes, Deckhouse only shows a warning.

To turn on fencing in the `master` node group, first add master nodes to the cluster. Then set the mode of the group.

Which network ports does fencing need?

The fencing agents on all nodes connect to each other on port 8500. Open the following access rules before you enable fencing:

- Port 8500 TCP and UDP between the InternalIP addresses of all cluster nodes.
- Port 8500 UDP between the addresses of the SystemNetwork on all nodes. This rule is necessary only if you use the second network (the `settings.fencing.systemNetworkFallback.systemNetworkName` parameter of the `node-manager` module). TCP is not necessary for this network. Currently the agents do not exchange messages through the second network. Open the rule in advance for the fallback through the SystemNetwork.

What does the D8FencingSystemNetworkNotFound alert mean?

The alert means that the `settings.fencing.systemNetworkFallback.systemNetworkName` parameter of the `node-manager` module names a SystemNetwork that does not exist in the cluster. The agents cannot use the second network. Deckhouse does not change the parameter.

To fix the problem, do one of the following:

- Create the SystemNetwork.
- Set the name of an existing network.
- Delete the name from the parameter.

Note the following:

- Deckhouse checks that the SystemNetwork exists when you set `systemNetworkName` to a new non-empty value. Deckhouse can reject only this change. Deckhouse always accepts the deletion of the value and the changes of the other ModuleConfig fields, also when the check is not available.
- During `dhctl bootstrap`, the check is not active. Deckhouse accepts the parameter without the check. If the network does not exist, the alert rises after the module starts.
- The alert stops about 10 minutes after you restore the network.
- While all node groups have the `Disabled` mode, the alert does not rise. The agents do not use the second network in this case. The alert rises when you enable fencing.

How does fencing protect the network key?

The fencing agents use a shared key to encrypt the messages between them. The key is in the `fencing-memberlist-key` Secret in the `d8-cloud-instance-manager` namespace. If you delete the Secret, Deckhouse generates a new key and the `D8FencingKeyRegenerated` alert rises.

When an agent gets a new key, it keeps the old key and starts to use the new key after 2 minutes. This delay lets the kubelet on all nodes update the Secret. The agent removes the old key after the same delay.

How do I rotate the fencing network key?

The `fencing-memberlist-key` Secret in the `d8-cloud-instance-manager` namespace has two keys in the `data` field:

- `primary`: the agents use this key to encrypt the messages. It is necessary.
- `secondary`: the agents use this key only to decrypt the messages. It is optional.

Each key is exactly 32 random bytes. In the `data` field, the value is these 32 bytes encoded in base64. Do not write a base64 string into the `stringData` field: the agents then get a key of a wrong length.

To rotate the key, do the following steps:

1. Make a new key, write it into `primary` and move the current key to `secondary`. The key stays in a shell variable only, so it does not appear in the output and in the shell history:

   ```
   NEW_KEY=$(head -c 32 /dev/urandom | base64)
   d8 k -n d8-cloud-instance-manager patch secret fencing-memberlist-key --type=json -p "
   [{\"op\":\"copy\",\"from\":\"/data/primary\",\"path\":\"/data/secondary\"},
   {\"op\":\"replace\",\"path\":\"/data/primary\",\"value\":\"${NEW_KEY}\"}]"
   unset NEW_KEY
   ```

   The kubelet updates the Secret on the nodes, usually in approximately 1 minute. Each agent keeps the current key, adds the new key and starts to use the new key after 2 minutes. During the rotation, the agents with the old key and the agents with the new key decrypt the messages of each other.

2. Wait at least 10 minutes. Then all agents use the new key. Remove the old key from the Secret:

   ```
   d8 k -n d8-cloud-instance-manager patch secret fencing-memberlist-key --type=json -p '[
     {"op": "remove", "path": "/data/secondary"}
   ]'
   ```

Do not delete the Secret to rotate the key. Deckhouse then makes a new key without the old key, and the `D8FencingKeyRegenerated` alert rises.

If the key has a wrong format, the agents that run continue to use their current keys and ignore the wrong key. The `d8_fencing_agent config_invalid{parameter="key"}` metric of these agents is `1`. An agent that starts with a wrong `primary` key cannot connect to the other agents and does not arm the watchdog. The `Fencing` condition of its node is `False` with the `InvalidKey` reason, and after 10 minutes the `D8FencingAgentInvalidKey` alert rises. To find these nodes, use the command from How do I check the fencing state of the nodes?. To check the length of the `primary` key, run the following command. It must show `32`:

```
d8 k -n d8-cloud-instance-manager get secret fencing-memberlist-key -o jsonpath='{.data.primary}'
| base64 -d | wc -c
```

How the fencing mechanism handles different node types?

The fencing mechanism stops a node that lost the connection with the cluster. It protects the cluster from “zombie” nodes in an undefined state.

A node with an armed watchdog works only while it sees the control plane. When the node does not see the control plane for longer than `controlPlaneLostAfter` of the profile, the fencing-agent stops the reset of the watchdog timer (the `softdog` kernel module) at once. When the timeout expires, a kernel panic occurs. This guarantees that no workloads remain running on the node.

While the node has a FencingFailedNodeState object, the node sees the control plane only when it can write its heartbeat through the API server. A node that cannot write its heartbeat stops in the same way.

After the agent stops the reset of the timer, it continues to check the control plane:

- If the control plane is available again before the watchdog timeout expires, the agent resets the timer again, and the node continues to work. Thus, a short pause of the control plane does not stop the nodes.
- The agent also asks the agents of the other nodes. It looks for a witness, that is, a node that sees the control plane. The agent sends a stop message to the witness, and the witness writes an attestation into the FencingFailedNodeState object of the node. After this message, the stop is final: the agent does not reset the timer again. The stop is also final when the FencingFailedNodeState object of the node is in the `Proven` phase.

A pause of the control plane in the whole cluster affects all nodes with an armed watchdog at the same time. Measurements on a cluster show that all nodes survive a pause of up to the following time:

Mode	Pause that the nodes survive
`Critical`	7.35 s
`Normal`	21.6 s
`Slow`	45.7 s

If the pause is longer, some or all nodes with an armed watchdog can stop.

For more details on how the fencing-agent works, refer to the description of the `spec.fencing.mode` parameter of the NodeGroup resource.

After the node stops, the behavior depends on the type of the node group (the `node.deckhouse.io/type` label) and its engine (`status.engine`):

1. The node stays down. Deckhouse disables the automatic reboot after a kernel panic at the OS level. This applies to all nodes of the groups with a fencing mode other than `Disabled`. This prevents a node from rejoining the cluster in an undefined state.
2. When the stop of the node is certain, the fencing-controller sets the `node.kubernetes.io/out-of-service` taint on the Node. The stop is certain when a witness confirmed it and the watchdog timeout of the node passed, or when the node is silent for the time that it needs to stop. For the times and for the cases when the controller does not set the taint, refer to When the controller sets the taint. After the taint, the controller sets the `Failed` phase and the `DisruptionTarget` condition on the pods of the node and deletes the pods immediately. This includes the pods that tolerate the taint. The controller does not delete the pods of DaemonSets and the static pods. The controller does not remove the finalizers of the pods: a pod with a finalizer stays in the API until the owner of the finalizer removes it. Pods of a StatefulSet and pods with ReadWriteOnce volumes start on other nodes. The evacuation is complete when the node has no `VolumeAttachment` objects. If the evacuation is not complete after 60 seconds, the incident stays in the `Evicting` phase and gets the `Progressing` condition with the status `False`. The reason of the condition tells what stops the evacuation now: `VolumesAttached`, `PodsRemaining` or `TooManyUnhealthyMachines`. The controller continues to examine the node and completes the evacuation when the cause is gone.
3. After the evacuation, Deckhouse acts on the node as follows:
   - For a node of a `CloudPermanent`, `CloudStatic`, or `Static` group (this includes a `Static` group with `staticInstances`, that is, CAPS), for a master node, and for an etcd arbiter node, Deckhouse only keeps the taint. It never deletes the Node object.
   - For a node of a `CloudEphemeral` group with `status.engine: CAPI`, Deckhouse deletes the Node object only when the MachineHealthCheck of the Cluster API cluster reports its machines and 50 percent or fewer of them are unhealthy. Deckhouse counts the stopped machine as unhealthy. MachineHealthCheck then sets the `HealthCheckSucceeded` condition of the Machine to `False` with the `NodeDeleted` reason and replaces the machine. If more than 50 percent of the machines are unhealthy, or the MachineHealthCheck does not report its machines yet, Deckhouse does not delete the Node object yet. The controller continues to examine the machines. When 50 percent or fewer of them are unhealthy, the controller deletes the Node object and completes the evacuation.
   - For a node of a `CloudEphemeral` group with `status.engine: MCM`, Deckhouse deletes the `Machine` (`machine.sapcloud.io`). Machine Controller Manager creates a new machine.
4. A node that Deckhouse does not replace stays down until you start it. When the node returns (the Node is `Ready` and it has a new boot ID), the fencing-controller removes the taint automatically. The controller works only while at least one node group has a mode other than `Disabled`. If you turn fencing off in all node groups, the taint stays on the node. Refer to What to do with the out-of-service taint before you turn fencing off in all node groups?

If the fencing-controller does not set the taint, Kubernetes sets the `node.kubernetes.io/unreachable` taint on the Node and evicts the pods of the node after the time in the `failedNodePodEvictionTimeoutSeconds` parameter of the `control-plane-manager` module (300 seconds by default).

A node of a group with fencing can return to the cluster in a way that is different from the behavior above. If this occurs, contact the support. Your report helps to clarify the sequence of checks and protection mechanisms.

When the controller sets the taint

The controller sets the `node.kubernetes.io/out-of-service` taint only on a node that is stopped. The controller proves the stop in one of two ways:

- Attestation of a witness. The witness is a node whose agent received the stop message of the

failed node. The agent of the witness writes the attestation into the FencingFailedNodeState object of the failed node. After the controller sees the attestation, it waits for `watchdog.timeout` + `watchdog.margin` of the profile. During this time, the watchdog of the failed node expires.
- Silence of the node. For example, the node is off or lost all networks, and no witness can confirm its stop. The controller waits for `controlPlaneLostAfter` + 0.2 s + `watchdog.timeout` + `watchdog.margin` + 0.9 s of the profile after the last sign of life of the node. The last sign of life is the later of two moments: when the controller saw the FencingFailedNodeState object first, and when it saw the last heartbeat of the node plus `heartbeat.interval`. During this time, the node detects the loss of the control plane and its watchdog expires.

The controller uses the way that gives the proof first:

Mode	Profile	Time after the attestation	Time after the last sign of life
`Critical`	`critical`	5.3 s	10.55 s
`Normal`	`normal`	15.75 s	25.95 s
`Slow`	`slow`	31.5 s	52.8 s

When the incident gets the `Proven` phase, the controller writes an event. The event tells the way of the proof and the time.

The controller takes the profile from the `spec.profileRef.name` field of the FencingFailedNodeState object. The agent that creates the object writes the slower of two profiles: the profile that the failed node applied, and the profile of the node group at the time of the creation of the object. Deckhouse does not let you change the mode of the group while a node of the group has a FencingFailedNodeState object. Refer to Known limitations of the evacuation.

The controller does not set the taint in these cases:

- The watchdog of the node was not armed (`spec.armed` is `false`). For example, the node is in a group with the `Disabled` mode.
- The `Fencing` condition of the Node is not `True`, that is, the agent of the node did not report that the watchdog is armed. The controller does not prove the failure, and the incident stays in the `Suspected` phase. The controller examines the node again every 5 seconds.
- The node sends heartbeats. The controller counts the time after the last sign of life again from each new heartbeat. A heartbeat after the attestation also cancels the proof by this attestation, and the `D8FencingNodeAliveAfterAttestation` alert rises.
- The node has one of the maintenance annotations `node-manager.deckhouse.io/fencing-disable`, `update.node.deckhouse.io/approved` or `update.node.deckhouse.io/disruption-approved`, or the agent of the node stops by plan. The controller examines the node again every 5 seconds.
- The owner reference of the FencingFailedNodeState object does not have the UID of the node with this name, for example after the node is created again with the same name. The controller does nothing for such an object.
- The controller cannot use the profile. The `D8FencingProfileIsUnavailable` alert rises.

Phases of a fencing incident

The controller writes the phase of the incident into the `status.phase` field of the FencingFailedNodeState object. The name of the object is the name of the node. To see the incidents, run the following command:

```
d8 k get fencingfailednodestates
```

The phases:

- `Suspected`: an agent found the failure of the node. The controller waits for the proof of the failure.
- `FallbackAlive`: the node sends its heartbeat through the API server. The node works, but the agents of other nodes cannot connect to it.
- `Proven`: the controller proved the failure by the attestation of a witness or by the silence of the node. Refer to When the controller sets the taint.
- `Evicting`: the node has the taint. The controller deletes the pods and waits until the node has no `VolumeAttachment` objects.
- `Evicted`: the evacuation is complete. The controller writes this phase first. Then it deletes the Node or the `Machine` of a node that the cloud replaces.

For the description of all fields, refer to FencingFailedNodeState.

Fencing alerts

The fencing-controller gives these alerts:

- `D8FencingNodeFailed`: a node with an armed watchdog has an incident.
- `D8FencingNodeIsolatedFromPeers`: the incident is in the `FallbackAlive` phase for 1 minute. The node works, but the network between this node and the other nodes is broken. Usually a firewall rule or an ACL stops the traffic of the agents.
- `D8FencingNodeAliveAfterAttestation`: the node sent a heartbeat after the attestation. The controller does not set the taint on this node while the node sends heartbeats. Examine the node.
- `D8FencingEvictionNotProgressing`: the evacuation is not complete after 60 seconds. The `reason` label tells the reason of the `Progressing` condition.
- `D8NodeOutOfServiceTainted`: a node has the `node.kubernetes.io/out-of-service` taint for 30 minutes. Start the node again or remove it from the cluster.
- `D8FencingProfileIsUnavailable`: the controller cannot use the profile of the incident and does not set the taint.

The alerts of the controller exist only while the controller works, that is, while at least one node group has a mode other than `Disabled`.

The `D8FencingWatchdogDeviceMissing` alert of the fencing agent rises when the agent has no watchdog device for 30 minutes. The agent does not arm the watchdog, so the controller never sets the taint on this node. Usually the DaemonSet of the node group did not replace the agent pod of the previous revision yet.

Known limitations of the evacuation

The fencing-controller runs on the master nodes. In a cluster with several master nodes, the controller has several replicas, and only the leader of the replicas works. A new leader counts the times of the incidents again from what it sees. Thus, if the leader changes during an incident, the taint comes later by up to approximately 20 seconds plus the time after the last sign of life from the table in When the controller sets the taint. No action is necessary. The leader can change in these cases:

- The master node of the leader fails.
- kubernetes-api-proxy restarts with a new manifest on the master node of the leader, for example, during an update of Deckhouse. The controller connects to the API server through this proxy.

On the master nodes of Deckhouse Engine (`systemType: Immutable`), and when the `control-plane-manager` module is disabled, the controller connects to the API server through the ClusterIP of the `kubernetes` Service. When a master node fails, the requests of the controller can go to the failed master node for some seconds. Then the taint of the failed node comes later, and the leader can change.

When the master node of the controller fails, the controller starts on another master node only after Kubernetes marks the failed master node as unreachable. Until then, the controller does not set the taint. Refer to How does fencing work on master nodes?

Deckhouse does not let you change the mode of a node group while a node of the group has a FencingFailedNodeState object. The controller calculates the times of an incident from the profile in the object, but the watchdog of the node can already use the times of the new mode. The error message names the nodes that have the objects. Wait until the incidents end, then change the mode. An incident ends when the node comes back after a reboot, or when its Node object is deleted. To find the objects:

```
d8 k get fencingfailednodestates
```

Change the mode of a node group only when all nodes of the group work correctly.

After switching the edition to CE, a FencingFailedNodeState object with the `critical` profile stays in the cluster, because CE has no such profile. The object gets the `ConfigurationError` condition, the `D8FencingProfileIsUnavailable` alert rises, and the controller does not evacuate the node. When the node comes back, that is, the Node is `Ready` and has a new boot ID, the controller deletes the object. In other cases, make sure that the node works or is off. Then delete the object:

```
d8 k delete fencingfailednodestate <NODE_NAME>
```

If the node fails again, the agents create a new object with the profile of the current mode. If the node has the `node.kubernetes.io/out-of-service` taint, the taint stays after the deletion. Remove the taint when the node works again:

```
d8 k taint node <NODE_NAME> node.kubernetes.io/out-of-service-
```

A node with an armed watchdog stops when it does not see the control plane. If the network separates

a part of the cluster from the control plane, all nodes with an armed watchdog in this part stop, also when they work correctly.

The controller knows that the watchdog of a node is armed only from the `Fencing` condition of the Node. The agent writes this condition after it arms or disarms the watchdog. Between these two moments, the condition can be incorrect. This gives two known cases:

- The agent armed the watchdog, but the `True` status of the condition did not reach the API server, and at this moment the node failed. For example, this can occur when the node fails during the restart of the agent. The controller does not set the taint on this node until the node comes back.
- The agent crashed, and the new agent process disarmed the watchdog. Then the node lost the control plane before the agent wrote the `False` status of the condition. The node works and its watchdog is not armed, but the condition stays `True`. The controller does not set the taint on this node: if the node gets a FencingFailedNodeState object, the object has `spec.armed: false`.

A rollback of Deckhouse to a version with fencing 1.0 is not supported. If you do a rollback, unload the `softdog` kernel module on the nodes. If you do not, the module keeps the timeout of fencing 2.0 (5 seconds for the `Critical` profile), the agent of fencing 1.0 resets the timer less frequently, and the node goes into a kernel panic.

What to do with the out-of-service taint before you turn fencing off in all node groups?

Only the fencing-controller removes the `node.kubernetes.io/out-of-service` taint from a node that came back. When the last node group gets the `Disabled` mode, Deckhouse deletes the controller together with the fencing agents. If a node has the taint at this time, the taint stays on the node. Deckhouse does not remove it, and no fencing alert rises after that. Kubernetes does not run pods on this node.

The same occurs when you disable the `node-manager` module, because Deckhouse deletes the controller together with the module. Deckhouse disables the module only after you confirm it with the `modules.deckhouse.io/allow-disabling=true` annotation on the ModuleConfig. Before you confirm, set the `Disabled` mode in all node groups and wait until no `fencing-agent` pods are left in the `d8-cloud-instance-manager` namespace. If you do not, an agent can fail to report its stop, and the watchdog stops its node. Also do the steps below before you set the `Disabled` mode in the last node group.

Before you set the `Disabled` mode in the last node group, do these steps:

1. Find the nodes with the taint and the fencing incidents:

   ```
   d8 k get nodes -o custom-columns='NAME:.metadata.name,TAINTS:.spec.taints[*].key' |
   grep -E '^NAME|node.kubernetes.io/out-of-service'
   d8 k get fencingfailednodestates
   ```

2. For each node with the taint, make sure that the node works again or that you replaced it. Then remove the taint:

   ```
   d8 k taint node <NODE_NAME> node.kubernetes.io/out-of-service-
   ```

   If the node is still stopped, you can turn fencing off. But then you must remove the taint manually after you start the node.

If you turn fencing on again, the controller removes the taint by the usual rule. It removes the taint when the Node is `Ready`, has a new boot ID and still has its FencingFailedNodeState object.

How do I work with GPU nodes?

GPU nodes are managed by the `gpu` module. The `node-manager` module does not prepare GPU nodes and does not deploy NVIDIA components.

The `spec.gpu` section of a NodeGroup is used by the `gpu` module: it selects the node groups that the module serves in Device Plugin mode and sets the GPU sharing strategy for them. If the `gpu` module is disabled, the `spec.gpu` section has no effect and the `NodeGroupGPUModuleDisabled` alert fires.

How to add GPU nodes, install the driver and NVIDIA Container Toolkit, configure sharing and MIG, and monitor GPUs is described in the `gpu` module documentation.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Managing nodes: examples

> The module lifecycle stage: General Availability

Below are some examples of NodeGroup description, as well as installing the cert-manager plugin for `kubectl` and setting the `sysctl` parameter.

Examples of the `NodeGroup` configuration

Cloud nodes

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
```

Static nodes

Use `nodeType: Static` for physical servers and VMs on Hypervisors.

An example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Adding nodes to such a group is done manually using the pre-made scripts.

You can also use a method that adds static nodes using the Cluster API Provider Static.

System nodes

Below is an example of a system node group manifest.

When describing a NodeGroup with Static nodes, specify the value `Static` in the `nodeType` field and use the `staticInstances` field to describe the parameters for provisioning static machines to the

cluster.

When describing a NodeGroup with CloudEphemeral type cloud nodes, specify the value `CloudEphemeral` in the `nodeType` field and use the `cloudInstances` field to describe the parameters for provisioning the cloud-based VMs.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  # Example for Static nodes.
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: system
  # Example for CloudEphemeral nodes.
  # nodeType: CloudEphemeral
  # cloudInstances:
  #   classReference:
  #     kind: YandexInstanceClass
  #     name: large
  #   maxPerZone: 2
  #   minPerZone: 1
  #   zones:
  #   - ru-central1-d
```

Nodes with GPU

GPU nodes are managed by the `gpu` module. The `spec.gpu` section of a NodeGroup takes effect only when the `gpu` module is enabled.

Examples of NodeGroup manifests for the Exclusive, TimeSlicing and MIG modes and a test workload are provided in the `gpu` module documentation.

Adding a static node to a cluster

Adding a static node can be done manually or using the Cluster API Provider Static.

Manually

Follow the steps below to add a new static node (e.g., VM or bare metal server) to the cluster:

1. For CloudStatic nodes in the following cloud providers, refer to the steps outlined in the documentation:
   - For AWS
   - For GCP
   - For YC
2. Use the existing one or create a new NodeGroup custom resource (see the example for the `NodeGroup` called `worker`). The nodeType parameter for static nodes in the NodeGroup must be `Static` or `CloudStatic`.
3. Get the Base64-encoded script code to add and configure the node.

   Here is how you can get Base64-encoded script code to add a node to the `worker` NodeGroup:

```
NODE_GROUP=worker
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} -o json | jq
'.data."bootstrap.sh"' -r
```

4. Pre-configure the new node according to the specifics of your environment. For example:
   - Add all the necessary mount points to the `/etc/fstab` file (NFS, Ceph, etc.).
   - Install the necessary packages.
   - Configure network connectivity between the new node and the other nodes of the cluster.
5. Connect to the new node over SSH and run the following command, inserting the Base64 string you got in step 3:

```
echo <Base64-CODE> | base64 -d | bash
```

Using the Cluster API Provider Static

⚠ If you have previously increased the number of master nodes in the cluster in the NodeGroup `master` (parameter `spec.staticInstances.count`), before adding regular nodes using CAPS, make sure that they will not be “captured”.

A brief example of adding a static node to a cluster using Cluster API Provider Static (CAPS):

1. Prepare the necessary resources.
   - Allocate a server (or a virtual machine), configure networking, etc. If required, install specific OS packages and add the mount points on the node.
   - Create a user (`caps` in the example below) and add it to sudoers by running the following command **on the server**:

     ```
     useradd -m -s /bin/bash caps
     usermod -aG sudo caps
     ```

   - Allow the user to run sudo commands without having to enter a password. For this, add the following line to the sudo configuration **on the server** (you can either edit the `/etc/sudoers` file, or run the `sudo visudo` command, or use some other method):

     ```
     caps ALL=(ALL) NOPASSWD: ALL
     ```

   - Generate a pair of SSH keys with an empty passphrase **on the server**:

     ```
     ssh-keygen -t  -f caps-id -C "" -N ""
     ```

     The public and private keys of the `caps` user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.
   - Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory **on the server**:

     ```
     mkdir -p /home/caps/.ssh
     cat caps-id.pub >> /home/caps/.ssh/authorized_keys
     chmod 700 /home/caps/.ssh
     chmod 600 /home/caps/.ssh/authorized_keys
     chown -R caps:caps /home/caps/
     ```

2. Create a SSHCredentials resource in the cluster:

   Run the following command in the user key directory **on the server** to encode the private key to Base64:

   ```
   base64 -w0 caps-id
   ```

   On any computer with `kubectl` configured to manage the cluster, create an environment variable

with the value of the Base64-encoded private key you generated in the previous step:

```
CAPS_PRIVATE_KEY_BASE64=<BASE64-ENCODED PRIVATE KEY>
```

Create a `SSHCredentials` resource in the cluster (note that from this point on, you have to use `kubectl` configured to manage the cluster):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: SSHCredentials
metadata:
  name: credentials
spec:
  user: caps
  privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
EOF
```

3. Create a StaticInstance resource in the cluster; specify the IP address of the static node server:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

4. Create a NodeGroup resource in the cluster. Value of `count` defines number of `staticInstances` which fall under the `labelSelector` that will be bootstrapped and joined into the `nodeGroup`, in this example this is `1`:

> **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new `NodeGroup` and move the static nodes into it by changing their labels.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

> **i** If it is necessary to add nodes to an existing node group, specify the desired number in the `.spec.count` field of the NodeGroup.

Using Cluster API Provider Static for multiple node groups

This example shows how you can use filters in the StaticInstance label selector to group static nodes and use them in different NodeGroups. Here, two node groups (`front` and `worker`) are used for different tasks. Each group includes nodes with different characteristics — the `front` group has two servers and the `worker` group has one.

1. Prepare the required resources (3 servers or virtual machines) and create the `SSHCredentials` resource in the same way as step 1 and step 2 of the example.
2. Create two NodeGroup in the cluster (from this point on, use `kubectl` configured to manage the cluster):

   > **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new `NodeGroup` and move the static nodes into it by changing their labels.

   ```
   d8 k create -f - <<EOF
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: front
   spec:
     nodeType: Static
     staticInstances:
       count: 2
       labelSelector:
         matchLabels:
           role: front
   ---
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     nodeType: Static
     staticInstances:
       count: 1
       labelSelector:
         matchLabels:
           role: worker
   EOF
   ```

3. Create StaticInstance resources in the cluster and specify the valid IP addresses of the servers:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

Cluster API Provider Static: Moving Instances Between Node Groups

⚠ During the process of transferring instances between node groups, the instance will be cleaned and re-bootstrapped, and the `Node` object will be recreated.

This section describes the process of moving static instances between different node groups (NodeGroup) using the Cluster API Provider Static (CAPS). The process involves modifying the NodeGroup configuration and updating the labels of the corresponding StaticInstance.

Initial Configuration

Assume that there is already a NodeGroup named `worker` in the cluster, configured to manage one static instance with the label `role: worker`.

`NodeGroup` worker:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
```

`StaticInstance` static-worker-1:

```
apiVersion: deckhouse.io/v1alpha2
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "192.168.1.100"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

Steps to Move an Instance Between Node Groups

1. Create a New `NodeGroup` for the Target Node Group

Create a new NodeGroup resource, for example, named `front`, which will manage a static instance with the label `role: front`.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: front
EOF
```

2. Update the Label on the `StaticInstance`

Change the `role` label of the existing StaticInstance from `worker` to `front`. This will allow the new NodeGroup `front` to manage this instance.

```
d8 k label staticinstance static-worker-1 role=front --overwrite
```

3. Decrease the Number of Static Instances in the Original `NodeGroup`

Update the NodeGroup resource `worker` by reducing the `count` parameter from `1` to `0`.

```
d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --type=merge
```

An example of the `NodeUser` configuration

```
apiVersion: deckhouse.io/v1
kind: NodeUser
metadata:
  name: testuser
spec:
  uid: 1100
  sshPublicKeys:
    - "<SSH_PUBLIC_KEY>"
  passwordHash: <PASSWORD_HASH>
  isSudoer: true
```

An example of the `NodeGroupConfiguration` configuration

Installing the cert-manager plugin for kubectl on master nodes

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-
manager/releases/download/v1.7.1/kubectl-cert_manager-linux-amd64.tar.gz -o - | tar
-zxvf - kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Tuning sysctl parameters

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "*"
  content: |
    sysctl -w vm.max_map_count=262144
```

Adding a root certificate to the host

⚠ Example is given for Ubuntu OS.
The method of adding certificates to the store may differ depending on the OS.

Change the bundles and content parameters to adapt the script to a different OS.

⚠ To use the certificate in `containerd` (including pulling containers from a private repository), a restart of the service is required after adding the certificate.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    JTEjMCEGA1UEAxMabmV4dXMuNTEuMjUwLjQxLjIuc3NsaXAuaW8wHhcNMjQwODAx
    MTAzMjA4WhcNMjQxMDMwMTAzMjA4WjAlMSMwIQYDVQQDExpuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pbzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL1p
    WLPr2c4SZX/i4IS59Ly1USPjRE21G4pMYewUjkSXnYv7hUkHvbNL/P9dmGBm2Jsl
    WFlRZbzCv7+5/J+9mPVL2TdTbWuAcTUyaG5GZ/1w64AmAWxqGMFx4eyD1zo9eSmN
    G2jis8VofL9dWDfUYhRzJ90qKxgK6k7tfhL0pv7IHDbqf28fCEnkvxsA98lGkq3H
    fUfvHV6Oi8pcyPZ/c8ayIf4+JOnf7oW/TgWqI7x6R1CkdzwepJ8oU7PGc0ySUWaP
    G5bH3ofBavL0bNEsyScz4TFCJ9b4aO5GFAOmgjFMMUi9qXDH72sBSrgi08Dxmimg
    Hfs198SZr3br5GTJoAkCAwEAAaN1MHMwDgYDVR0PAQH/BAQDAgWgMAwGA1UdEwEB
    /wQCMAAwUwYDVR0RBEwwSoIPbmV4dXMuc3ZjLmxvY2FsghpuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pb4IbZG9ja2VyLjUxLjI1MC40MS4yLnNzbGlwLmlvMA0GCSqG
    SIb3DQEBCwUAA4IBAQBvTjTTXWeWtfaUDrcp1YW1pKgZ7lTb27f3QCxukXpbC+wL
    dcb4EP/vDf+UqCogKl6rCEA0i23Dtn85KAE9PQZFfI5hLulptdOgUhO3Udluoy36
    D4WvUoCfgPgx12FrdanQBBja+oDsT1QeOpKwQJuwjpZcGfB2YZqhO0UcJpC8kxtU
    by3uoxJoveHPRlbM2+ACPBPlHu/yH7st24sr1CodJHNt6P8ugIBAZxi3/Hq0wj4K
    aaQzdGXeFckWaxIny7F1M3cIWEXWzhAFnoTgrwlklf7N7VWHPIvlIh1EYASsVYKn
    iATq8C7qhUOGsknDh3QSpOJeJmpcBwln11/9BGRP
    -----END CERTIFICATE-----
    EOF
    )

    # bb-event           - Creating subscription for event function. More
information: http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to
the store
      update-ca-certificates
    }

    # bb-tmp-file - Creating temp file function. More information:
http://www.bashbooster.net/#tmp
    CERT_TMP_FILE="$( bb-tmp-file )"
```

```
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    # bb-sync-file                                  - File synchronization function.
More information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"     - Destination file
    ##  ${CERT_TMP_FILE}                           - Source file
    ##  ca-file-updated                            - Name of event that will be called
if the file changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated
```

Adding the ability to download images from insecure container registry to containerd

The ability to download images from an insecure container registry is enabled using the `insecure_skip_verify` parameter in the containerd configuration file. For more information, see the How to add configuration for an additional registry.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The okmeter module

> The module lifecycle stage: General Availability

The `okmeter` module installs the Okmeter agent as a `daemonset` in the `d8-okmeter` namespace, and deletes `okmeter` installed manually.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

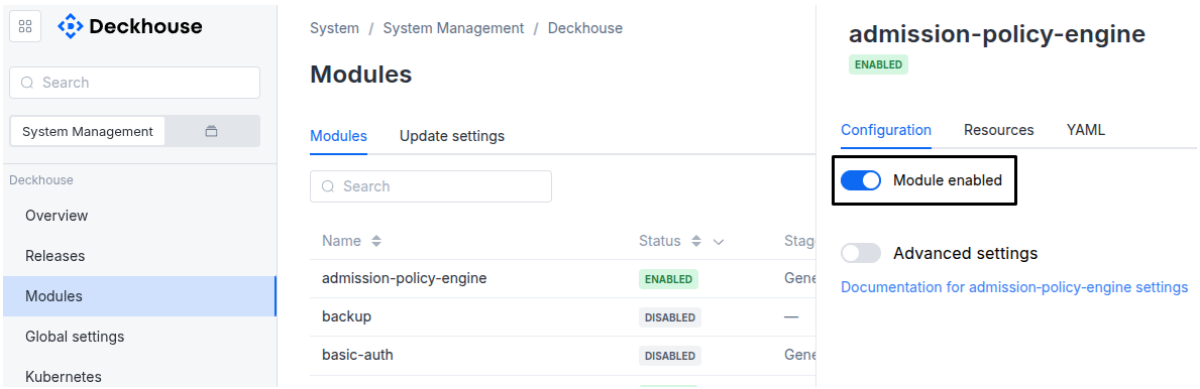
The module has 1 alert.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `okmeter` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable okmeter
  ```

- **Using ModuleConfig `okmeter`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `okmeter` (create it if necessary);

  Example of a manifest to enable module `okmeter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: okmeter
  spec:
    enabled: true
  ```

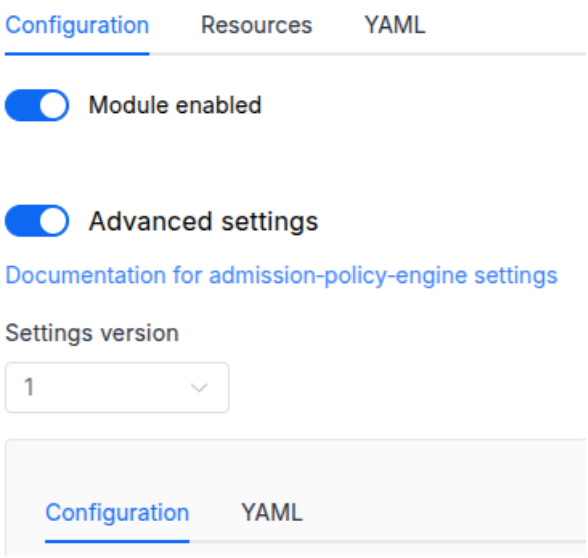
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `okmeter` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `okmeter` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module

`okmeter`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `okmeter` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc okmeter
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `okmeter` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `okmeter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: okmeter
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.apiKey**
    string

    Required value

    Key to access Okmeter API.

    You can get the key from your personal Okmeter account (`OKMETER_API_TOKEN`).

    Example:

    ```
    apiKey: 5ff9z2a3-9127-1sh4-2192-06a3fc6e13e3
    ```

  - **settings.disableAutoDiscovery**

    Disables okagent autodiscovery. Setting this option to `true` will leave only basic host monitoring, all other plugins (postgresql/mysql/redis/etc.) will be disabled. Default value is `false`

  - **settings.image**
    object
    - **settings.image.repository**
      string

      Required value

      Docker repository for okmeter agent image.

      Default: `registry.okmeter.io/agent/okagent`

    - **settings.image.tag**
      string

Required value

Okmeter agent image tag.

Default: `latest`

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    test: test
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

The module lifecycle stage: General Availability

Example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  version: 1
  enabled: true
  settings:
    apiKey: dh-test-secret-0123456789abcdef
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The openvpn module

> The module lifecycle stage: General Availability

> Web interfaces associated with the module: openvpn-admin

The openvpn module allows peers to authenticate each other using certificates and provides a simple web interface.

Using the web interface, you can:

- Issue certificates.
- Revoke certificates.
- Cancel certificate revocation.
- Get a ready-to-use custom configuration file.

Integrating with the user-authn module lets you manage user access to the web interface.

Exposing the VPN service

1. Select one or more external IP addresses for connection.
2. Use one of the connection methods:
   - By external IP address (`ExternalIP`) - if there are nodes with public IP addresses.
   - Using `LoadBalancer` - for all cloud providers and their placement schemes that support LoadBalancer ordering.
   - `Direct` - configure the traffic path manually: from the entry point to the cluster to the pod with OpenVPN.

Available cluster resources after connecting to the VPN

The following parameters are pushed to the user's computer after connecting to the VPN:

- The `kube-dns` address is added to the client's DNS servers to allow direct access to Kubernetes services via FQDN.
- A route to the local network.
- A route to the cluster service network.
- A route to the Pod network.

User's traffic audit

The module can log user's activity via VPN in JSON format. Traffic is grouped by `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto` fields. Using the log-shipper module, container logs can be collected and stored for later auditing.

Third-party components

List of third-party software used in the `openvpn` module:

- OpenVPN 2.6.12

  License: Mixed

  A secure tunneling daemon.

- OpenSSL 3.3.2

  License: Apache License 2.0

  TLS/SSL and crypto library.

- Libcap-ng 0.8.5

  License: GNU General Public License v2.0

  Libcap-ng is a library for Linux that makes using posix capabilities easy..

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

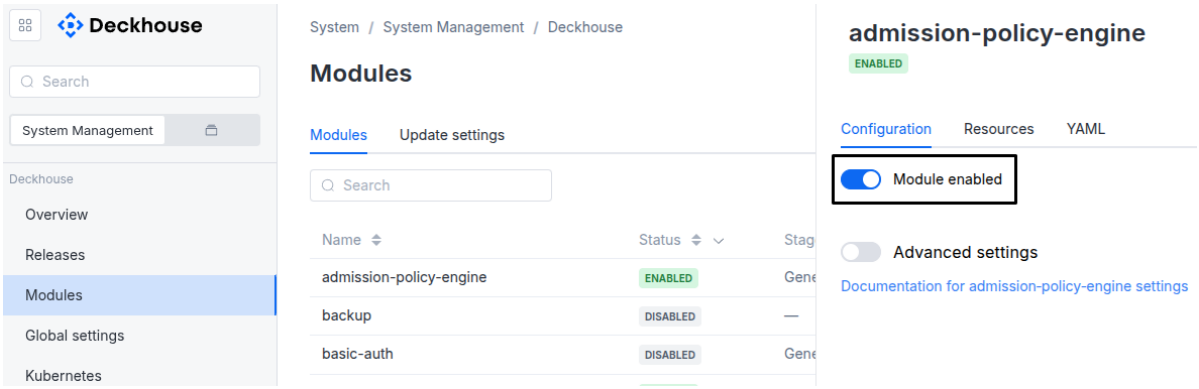
The module has 9 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `openvpn` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable openvpn
```

- **Using ModuleConfig `openvpn`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `openvpn` (create it if necessary);

  Example of a manifest to enable module `openvpn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  enabled: true
```

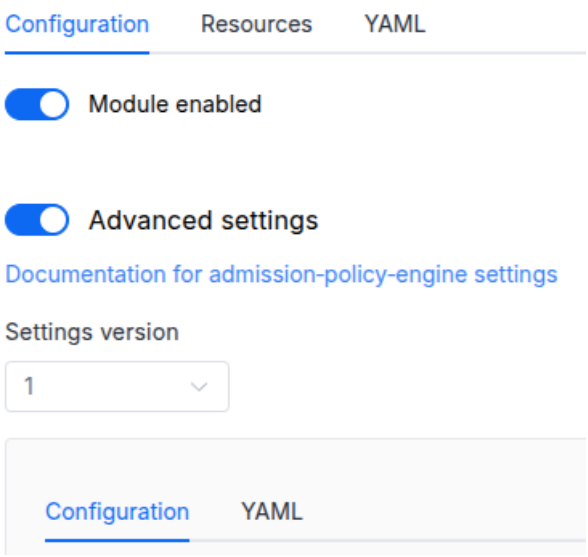
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `openvpn` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `openvpn` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `openvpn`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `openvpn` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc openvpn
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `openvpn` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `openvpn`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: openvpn
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic

transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

  - `del(.auth.password) | if .auth == {} then del(.auth) end`
  - `del(.storageClass)`

Parameters

Schema version: 2

- **settings**
  object
  - **settings.auth**
    object

    Options related to authentication or authorization in the application.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access the OpenVPN admin panel.

      This parameter is used if the user-authn module is enabled or the externalAuthentication parameter is set.

      > **Caution!** Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

      > **Note!** External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

      - **settings.auth.externalAuthentication.authURL**
        string

        The URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

    - **settings.auth.whitelistSourceRanges**
      array of strings

      The CIDR range for which authentication to access the OpenVPN is allowed.

  - **settings.clientCertExpirationDays**
    integer

    Certificate client expired period for OpenVPN in days. Specifies the number of days for which

a client certificate generated using EasyRSA will be issued.

Client certificate lifetime must be at least 1 day and no more than 10 years.

Default: `3650`

Allowed values: `1 <= X <= 3650`

- **settings.externalHost**
  string

  An IP address or a domain clients use to connect to the OpenVPN server.

  By default, data from an `openvpn-external` service are used.

- **settings.externalIP**
  string

  The IP address of a cluster node to connect OpenVPN clients.

  It is only required if the `ExternalIP` inlet is used.

- **settings.externalPort**
  integer

  The port to expose on the `externalIP` or load balancer.

  Default: `5416`

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

- **settings.hostPort**
  integer

  Port to connect to the OpenVPN server, which will be available on the node where it is scheduled.

  The parameter is available when selecting inlet `HostPort`.

  Default: `5416`

- **settings.https**
  object

  What certificate type to use with the OpenVPN admin panel.

  This parameter completely overrides the `global.modules.https` settings.

  Example:

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for the OpenVPN admin panel (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**
    object

- **settings.https.customCertificate.secretName**
  string

  The name of the Secret in the `d8-system` namespace to use with the OpenVPN admin panel (this Secret must have the kubernetes.io/tls format).

  Default: `false`

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — the OpenVPN admin panel will use HTTPS and get a certificate from the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — the OpenVPN admin panel will use the certificate from the `d8-system` namespace for HTTPS.
  - `Disabled` — in this mode, the OpenVPN admin panel works over HTTP only.
  - `OnlyInURI` — the OpenVPN admin panel will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the user-authn will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for the OpenVPN admin panel.

  By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.inlet**
  string

  The way the connection is implemented.

  The following inlet types are supported:

  - `ExternalIP` — when there are nodes with public IPs. It is used together with the `externalIP` parameter.
  - `LoadBalancer` — for all cloud providers and cloud-based placement strategies that support the provision of LoadBalancers.
  - `HostPort` — the port of the OpenVPN server will be available on the node where it is scheduled. The port can be configured in the `hostPort` parameter.
  - `Direct` — for non-standard cases. You need to create a service called `openvpn-external` in the `d8-openvpn` namespace. It will route traffic to the Pod with the `app: openvpn` label to the port called `ovpn-tcp` (or just 1194). This service provides the externalIP, the IP address of the balancer or its host. If none of these are present, you need to specify the `externalHost` parameter.

  Allowed values: `ExternalIP`, `LoadBalancer`, `HostPort`, `Direct`

- **settings.loadBalancer**
  object

  A section of optional parameters of the `LoadBalancer` inlet.

  - **settings.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

    > **Note** that module does not take into account the specifics of setting annotations in different clouds. If annotations for the provision of the load balancer are only used when the service is being created, then you need to restart the module (disable/enable it) to update them.

  - **settings.loadBalancer.sourceRanges**
    array of strings

    A list of CIDRs that are allowed to connect to the Load Balancer.

The cloud provider may not support this option or ignore it.

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.pmacctEnabled**
  boolean

  Enable logging of user's activity via VPN in JSON format.

  All connections within `tun` interfaces will be collected via `libpcap` and logged as `{"event_type": "purge", "ip_src": "172.25.175.10", "ip_dst": "10.222.0.10", "port_src": 32172, "port_dst": 53, "ip_proto": "udp", "packets": 1, "bytes": 53}`

  Default: `false`

- **settings.pushDefaultGatewayToClient**
  boolean

  Routing all client traffic through the VPN. A default route will be added to clients.

  Default: `false`

- **settings.pushToClientDNS**
  string

  The IP address of the DNS server to send to clients upon connection.

  By default, the IP address of the `kube-system/kube-dns` service is used.

- **settings.pushToClientRoutes**
  array of strings

  A list of routes to send to clients upon their connection.

  By default, this list is generated automatically using the local cluster network, service subnet, and Pod subnet.

  Example:

  ```
  pushToClientRoutes:
  - 172.25.176.0/24
  - 172.25.178.0/255.255.255.0
  ```

- **settings.pushToClientSearchDomains**
  array of strings

  A list of search domains to send to clients upon connection.

  The default value is a value from the `global.discovery.clusterDomain` variable.

- **settings.tcpEnabled**
  boolean

  Default: `true`

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**

string
- **settings.tunnelNetwork**
  string

  Tunnel network for OpenVPN server.

  Default: `172.25.175.0/24`

  Examples:

  ```
  tunnelNetwork: 172.25.175.0/24
  ```

  ```
  tunnelNetwork: 172.25.178.0/255.255.255.0
  ```
- **settings.udpEnabled**
  boolean

  Default: `false`

Authentication

user-authn module provides authentication by default. You can also configure authentication using the externalAuthentication parameter. If these options are disabled, the module will use basic auth with the auto-generated password.

To view the generated password, run the command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values openvpn -o json | jq '.openvpn.internal.auth.password'
```

To generate a new password, delete the Secret:

```
d8 k -n d8-openvpn delete secret/basic-auth
```

> The `auth.password` parameter is deprecated.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability

Why doesn’t automatic DNS server configuration work when connecting to macOS and Linux using the OpenVPN client

Due to the architectural features of the Linux and macOS operating systems, automatic DNS server configuration when using the official OpenVPN client is not possible.

To configure the DNS server on such operating systems, the service provides the ability to use third-party scripts that run when the client connects or disconnects.

In the client configurations generated by the module, the blocks responsible for these settings are predefined and commented out:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

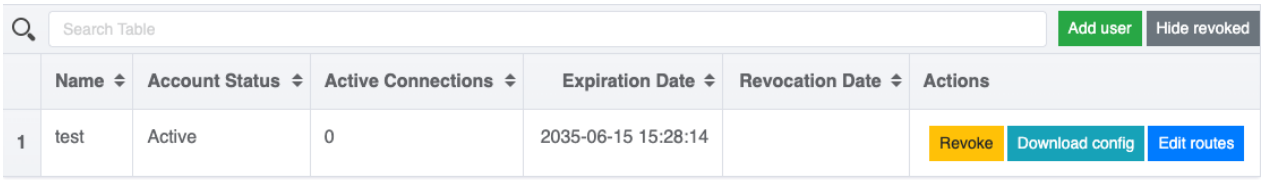
To use these code blocks, uncomment them (remove the initial `#` symbol), and specify the correct paths to the scripts.

You can either prepare the scripts yourself or use ready-made scripts from the official OpenVPN Community (for Linux). For macOS, you can use a third-party script.

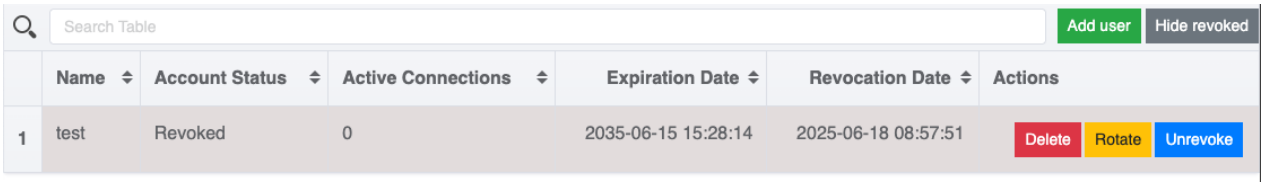
⚠ Scripts must have execution permissions.

How to revoke, rotate or delete a user certificate

All operations with client certificates are performed via the openvpn-admin web interface. Buttons for managing certificates are available to the right of each user's name:



To rotate (issue a new certificate) or delete a client, you must first revoke their current certificate:



Once revoked, the Renew (rotation) and Delete (removal) actions become available.

How to rotate a server certificate

The server certificate is rotated automatically several days before it expires.

If you need to rotate the certificate manually (e.g., due to certificate corruption or an unscheduled replacement), follow these steps:

1. Delete the secret `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-server
   ```

2. Restart the OpenVPN pods to trigger the generation of a new certificate:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

A new certificate will be generated automatically when the pods start.

How to rotate a root certificate (CA)

The root certificate (CA) and server certificate are rotated automatically 1 day before expiration. Automatic rotation of clients certificates is not provided. The root certificate (CA) is used to sign all certificates in OpenVPN — both server and client. Therefore, when replacing the CA, you must reissue all dependent certificates.

Steps to rotate the root certificate:

1. Revoke or delete all active client certificates using the openvpn-admin web interface. If you choose to revoke certificates, you can rotate them later (Renew) after the CA is replaced, without recreating the clients.
2. Delete secrets `openvpn-pki-ca` and `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
   ```

3. Restart OpenVPN pods:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

4. Rotate certificates of revoked users, or create new ones.
5. Delete all revoked certificate secrets:

   ```
   d8 k -n d8-openvpn delete secrets -l revokedForever=true
   ```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

> The module lifecycle stage: General Availability

An example for bare metal clusters

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
   inlet: ExternalIP
   externalIP: 5.4.54.4
```

An example for AWS & Google Cloud

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

An example for an external load balancer with a public IP address

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # The internal IP address to forward the external LB's
traffic to.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Registry Module

> Included in extensions: **Advanced Infrastructure Security, Billing**

> ⚠ The module lifecycle stage: Preview

The module is used to manage the pull paths for container images of Deckhouse Platform (DP) components, and for images from additional registries (vendor or your own).

The current implementation of the module and the v1 implementation, used in DP up to and including 1.77, are described below.

> A cluster where the v1 implementation of this module has never run uses the current one right away. A cluster where the v1 implementation has run keeps using it until it is switched to the `Unmanaged` state, after which the migration completes on its own. The process is described in How does the migration to the registry module work.

Current implementation

Modes of operation

The module can run in one of the following modes (selected with the `mode` parameter):

- `Unmanaged` (the default mode). In this mode the module does not manage the pull paths of DP components. With `mode: Unmanaged` the cluster keeps pulling images from the registry it was installed with (set during cluster bootstrap in the `deckhouse` parameter in InitConfiguration). None of the module's components are created in this mode.
- `Managed`. In this mode the module manages the pull paths for container images. The cluster administrator hands the pull path to the module by setting the `mode: Managed` parameter.

  In this mode (when the module manages the pull paths) the following independent settings are used:

  - `primary.upstream` — the registry images are pulled from. If not set, the cluster is considered air-gapped: the only source becomes the in-cluster cache, filled with the `d8 mirror push` command.
  - `storage.cache` — controls the in-cluster cache on the control-plane nodes. If set, a registry is deployed on the control-plane nodes. All nodes pull images from it, with the upstream staying a fallback path while the cache is being filled.

  Turning the cache on or off, changing the registry, changing credentials — each of these is an ordinary reconfiguration, possible at any time and in any order. Exactly one change has a condition attached: when the upstream is removed to go air-gapped, the module waits until the in-cluster cache holds the whole expected image set and no workload in the `kube-system` and `d8-*` namespaces references the upstream any more — only then do images stop being pulled from the registry set in `primary.upstream`. Workloads in other namespaces are not counted: move the ones whose images reference the upstream address onto the in-cluster address before removing the upstream. While this is in progress, the `D8RegistryAirGapTransitionHeld` alert fires. If the switch to the in-cluster cache happened right after the upstream was removed, every node could be left with nowhere to pull images from.

  The cache reclaims its own disk space. Nothing else ever removes data from it — every release adds a slice of the repository — so a sweep removes the slices of releases the cluster has already passed through, keeping the deployed release and the one before it. It runs on a schedule (nightly by default), once the cache holds a new image set — after a Deckhouse Platform update, or after a module is enabled or updated — and when the store refuses writes. A sweep does not interrupt

serving, but it pauses filling the replica's store and can fail a `d8 mirror push` started during it, which is why it defaults to a nighttime hour. It can also be run right away with a RegistryOperation object. For details, see the `storage.garbageCollection` parameter and the FAQ section The cache keeps growing. What reclaims it.

The `primary.upstream` and `storage.cache` parameters together cover every supported configuration:

primary.upstream	storage.cache	What the cluster does
set	`false`	Nodes pull images directly from the upstream. Nothing is deployed on the control-plane nodes
set	`true`	The cache acts as a pass-through: filled from the upstream on demand and ahead of time
not set	`true`	Air-gapped. The cache is the only source, and `d8 mirror push` is the only way in
not set	`false`	The configuration is rejected: nodes would have nowhere to pull images from

Installer behavior when no container image registry is configured

The installer creates the `registry` ModuleConfig with `mode: Managed` and `primary.upstream` pointing to the public Deckhouse Open/CE registry `registry.deckhouse.io/deckhouse/ce`, and warns about it in its output when the installation configuration specifies no container image registry in any of the following places:

- InitConfiguration
- The `registry` section of the `deckhouse` ModuleConfig
- The `registry` ModuleConfig (the ModuleConfig is absent)

This applies only when the installation configuration contains a ClusterConfiguration and the container runtime is managed by Deckhouse Platform (`defaultCRI` is `Containerd` or `ContainerdV2`). In a Managed Kubernetes cluster (without ClusterConfiguration) and with `defaultCRI: NotManaged`, the installer does not create this ModuleConfig.

Under the same conditions, the installer rejects a configuration in which the `registry` ModuleConfig sets `mode: Unmanaged` (or omits `mode`, which means the same) or `enabled: false` and no registry is specified elsewhere. In this case, the nodes would have nowhere to pull images from. Specify the registry from which nodes pull images directly in one of the following ways:

- Via the `deckhouse` parameter in InitConfiguration (`imagesRepo` and `registryDockerCfg`).
- By setting `mode: Managed` with `primary.upstream` in the `registry` ModuleConfig. A private or mirror registry populated with `d8 mirror push` can serve as `primary.upstream`. To install an air-gapped cluster from an image bundle instead, set `enabled: true` and `mode: Managed` with `storage.cache: true` and `storage.source`, without `primary.upstream`, and pass the bundle to the installer with the `--img-bundle-path` flag.

The installer also refuses a `registry` ModuleConfig that specifies `primary` or `storage` without `mode: Managed`, as the module's schema does: in the default `Unmanaged` mode these settings would have no effect.

Additional registries

`primary.upstream` is deliberately the only registry configurable through the module. It is used for the images of DP components. Additional registries (a vendor's, needed by some module, or your own) are declared as a separate RegistryUpstream resource. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryUpstream
metadata:
  name: virtualization-images
spec:
  match: images.virtualization.example.com
  upstream:
    host: vendor.example.com
    path: /virtualization
    auth:
      username: robot
      password: <PASSWORD>
```

This approach — declaring an additional registry as a separate resource — was chosen because a module or a user adding their own registry should not have to edit a ModuleConfig they don’t own. Such registries are always transit: pulls are routed through the node agent so that credentials and CA certificates live in one place, but nothing from them is ever cached.

Mirrors for remote nodes

Nodes far from the control plane, such as GPU workers in another region, pull every image from the in-cluster cache on the control-plane nodes or from the upstream. For large images this is slow, and the nodes wait for it. A RegistryMirror points a group of nodes at a registry near them that holds a copy of the image set, such as a Harbor proxy-cache project. You run and fill that registry yourself; the cluster only decides which nodes ask it, and for what.

The agent on a selected node asks the mirror for manifests and blobs named by digest only. Tags are resolved by the cache or the upstream as before, so which version runs stays the cluster’s decision, and the content the mirror serves is checked against its digest. If the mirror does not hold an image, refuses the request or does not answer, the pull goes to the cache or the upstream, as without a mirror. A node is served by one mirror at most.

A mirror takes effect in the `Managed` mode of the module. See the example.

Architecture

The node agent

On every node of the cluster, an agent (registry-agent) is deployed as a static pod. The container runtime is pointed at it (`127.0.0.1:5001`, through containerd’s `_default` fallback directory) and gets access to registries through it, passing the original registry name along in the request.

This achieves the following:

- **Node configuration is static.** Adding a registry, turning the in-cluster cache on, rotating credentials — none of this changes the configuration on any node, because the container runtime accesses registries through the agent.
- **Registry credentials stay out of the runtime’s configuration files.** They are held by the agent and attached to every request.
- **Credentials are not stored in the module’s resources.** A RegistryNode, a RegistryStorage, and the recorded effective upstream name a key in one Secret instead of carrying the credential itself. These are cluster-scoped resources, and the permission to read them is granted to every node, so a credential inside any of them would be readable through the API by every kubelet in the cluster. Each component’s access is narrowed to that one Secret by name, and the credential it resolved is kept together with its own copy of the routing.
- **Pulling images works when the API server is unavailable.** The agent keeps a copy of its routing on disk and falls back to it, because the images it serves include the ones needed to repair a broken control plane. A node that could not reach the API server uses the routing it was installed with.

An unconfigured registry is proxied untouched, along with whatever credentials the request already carried — so an ordinary `imagePullSecret` for a third-party registry keeps working, with nothing needing to be declared for it.

The in-cluster cache

With `storage.cache` enabled, a registry runs on each control-plane node. It stores blobs (the files an image is made of) in the cache directory: `/var/lib/deckhouse/registry` when the master NodeGroup

has `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` otherwise. One control-plane node holds the lease and fills from the upstream; the others replicate data from the leading replica (the one holding the lease) ahead of time, so losing the leader does not mean filling from scratch. For details on how the leading replica is chosen, see Leader election.

Agents reach a replica by the node's address, not by the service name `registry.d8-system.svc:5001`. `registry.d8-system.svc:5001` is what every image reference in the cluster is built from, and what a request is matched against. But the agent runs in the host's network namespace, where the cluster DNS name does not resolve.

A replica serves on port `4281/TCP` of its control-plane node: the agents of every node connect to it, and so do the other replicas when they replicate. The publication endpoint behind the Ingress is port `4283/TCP` of the same nodes, reached from the nodes running the ingress controller. Both are in the 4200-4299 range and are listed in "Network interaction of the platform components". The `5001` in the service name is not the cache's port: on every node, `127.0.0.1:5001` is the agent.

That address (which agents use to reach a replica) takes effect in two steps, and the order matters. Nodes are given it as soon as the module starts managing the pull path, through the same bashible rollout that installs the agent. The image references of the platform's own components move only after every node's agent has applied the configuration it was given: no node resolves this address except through the agent, so re-rendering the cluster onto it any earlier would point every workload at something that cannot be pulled yet. Until then, references keep naming the registry the cluster was installed with — which is also where they return if the module is set to `Unmanaged` or disabled. Whether this step has happened is visible from the presence of the ConfigMap `registry-image-address` in `d8-system`.

The Deckhouse controller pulls through the same agent — at `127.0.0.1:5001` — its own release channel and the module sources it owns. A different address, but the same agent: the controller is a process and has to dial a connection, whereas an image reference is resolved by the container runtime, which a drop-in redirects to the agent, so it can name a service that nothing ever connects to. This is exactly what makes changing the registry a single change: the configuration is edited in one place, the agents are reconfigured, and both what the nodes pull and what the controller fetches follow — with no registry address for the controller written down anywhere, and so none that could be left pointing at the old one.

> ⚠ Use a separate disk for the cache directory and for etcd data. Sharing one disk degrades etcd while the cache is being filled.

The expected image set

The cache holds the expected image set: the images of the deployed DP release and of the previous one (for a rollback), the images of the modules the cluster runs, and the catalog of modules the module sources offer. Updates the cluster lists as pending — of DP and of modules — are copied too, ahead of time: the replicas receive an update before it is deployed, in an air-gapped cluster as soon as `d8 mirror push` has put it into the store. An update the source does not hold yet does not hold the cache back; it is copied when it is deployed at the latest. The replicas copy only while one of them is short of the set or a pending update has appeared; after that, nothing is copied until the set changes again.

Replication

The followers copy from the leader's read port, the one the nodes pull through. A follower writes only into its own store, through its own write port, and no replica uses another replica's write port, so between control-plane nodes replication needs only the read port. The leader's write port still has to be reachable from the nodes that run the ingress controller: the publication endpoint forwards `d8 mirror push` to it (see Air-gapped clusters). What a follower can get from the leader's read port depends on whether an upstream is set:

- With an upstream, the leader's read port is a pull-through cache for the upstream's images: an image the leader does not hold yet, it fetches from the upstream to answer the follower, and keeps. A follower therefore does not wait for the leader to finish filling. An image that a replication pass still leaves missing (`pending_on_the_leader` in the syncer's `the replication finished` log line) is one the leader has not got yet: for example, a module from another module source, which the leader fills from that source's own registry rather than through the upstream, or an image the upstream could not serve at that moment. A later pass picks it up.
- Without an upstream, the leader serves only what it holds. An image missing on the leader is missing at the source: `d8 mirror push` has not brought it yet.

Only the copy ahead of pending updates is rate-limited: it reads from its source at no more than 16

MiB/s, so that it does not take a control-plane node's CPU away from etcd and the API server. The leader's fill of the expected set and the followers' replication of it are not limited: until a store holds the set, nodes pull through the upstream, and a follower without the set is a replica the cluster cannot fall back on (in an air-gapped cluster, there is nothing else to fall back on). The followers replicate independently of each other, so all of them read from the leader at the same time, each copying several layers at once, on top of the nodes' own pulls. None of this is configurable in the ModuleConfig.

Leader election

The leader fills the store and is the source the other replicas copy from, so the lease is not handed out by a plain election:

- When some replica holds the whole expected image set, a leader that cannot complete the set by itself steps aside for that replica. A leader cannot complete the set by itself when there is no upstream, when its last pass failed, when its store refuses writes, or when it has stopped renewing the lease.
- A leader that is still completing the set from the upstream keeps the lease even if a follower finishes first: the follower copies from the leader, and moving the lease would gain nothing.
- While no replica holds the whole set, as in a new cluster, the lease does not move: every move would start the fill over on another replica. If there is no leader at all, the lease goes to the replica that holds the most verified images; ties are broken by node name. A replica whose registry does not answer or whose disk refuses writes is passed over, and so is one whose store is out of room while another replica has room. A failed pass alone does not pass a replica over: what it holds is still counted from its store.

So an incomplete replica can be the leader, either while no replica is full or while it is filling from the upstream. The fields that show this are described in the FAQ.

The rule that a full replica takes over matters most in an air-gapped cluster. There the leader gets images only from `d8 mirror push`, which always lands on the leader, and the followers copy only what the leader holds. An incomplete leader beside a full follower (for example, a leader whose store was lost) could not complete the set, the followers would replicate its gaps, and the images on the full replica would reach nobody.

A replica whose node has been reported unreachable for a minute (about two minutes after the node stops responding) is not counted in the election, so the replicas that are left do not wait for a leader whose node is gone. A replica whose registry stops answering or whose disk stops accepting writes reports an error and hands the lease over to a replica that holds the full expected image set, if there is one; otherwise it keeps the lease and only reports the error.

After the leader's node fails, a new leader takes over in about a minute in an air-gapped cluster and in two to three minutes in a cluster with an upstream. While no replica holds the full expected image set yet, it can take up to about three and a half minutes, and two minutes more if the replica holding the most images does not take the lease. Nodes pull from the leader only, so until the new leader takes over and the nodes' configuration follows it, they pull from the upstream if one is configured; in an air-gapped cluster, pulls of images that are not already on the node wait for the new leader. The other replicas keep their data, so a new leader that holds the full expected image set fills nothing again.

While no replica holds the full expected image set, the replica that takes the lease may hold fewer images than another one, or none: the replica holding the most images is passed over if its node is unreachable or if it does not take the lease within two minutes. The new leader then fills what it lacks again, from the upstream, or, in an air-gapped cluster, from what is pushed again with `d8 mirror push`. It keeps the lease until some replica holds the full expected image set, even after the replica holding more images comes back.

A leader whose store has lost the expected image set, for example after the cache directory on its node was wiped, gives up the lease while another replica holds the whole set: it checks the deployed release and a sample of the set on its own disk about once a minute, and at startup. It reports `full: false` and `verifiedDigests: 0`, the replica that holds the set becomes the leader, and the former leader copies the set from it like any other replica. If no other replica holds the whole set, the leader keeps the lease, since it is still the one that fills the set, but reports `full: false` and `verifiedDigests: 0` all the same: `safeToDropUpstream` becomes `false`, and the leader fills the set again from the upstream (in an air-gapped cluster, counts its store again).

A replica whose store holds at most half as many manifests as its `verifiedDigests` (by `totalDigests`) stops reporting the set within about a minute and a half of the loss: the leader as described above, and a replica that does not hold the lease by reporting `full: false` and reading its store again. A smaller loss on a replica that does not hold the lease is reported no later than at its next periodic check of its store, every six hours by default. After the leader's syncer restarts, `safeToDropUpstream` stays `false` until the running syncer has read its store, normally at its first pass, within about two minutes, so a report left by the previous process cannot drop the upstream.

What protects the ports on control-plane nodes

The cache runs in the host network, so its ports are open on the address of every control-plane node, to anything that can reach that address: other nodes, pods, and hosts outside the cluster if the network lets them through. The module ships no host firewall policy for these ports (for example, Cilium host policies). Restricting who can reach a control-plane node at the network level is left to the cluster's own firewall. Each port protects itself as follows:

Port	Purpose	Protection
`4281/TCP`	Reading: the agents of all nodes pull through it, the other replicas replicate from it.	TLS with the module's own certificate authority. Every request needs a token from the module's token service. The read credentials, which every node uses to get one, are on every node, so whoever has taken them from any node and can reach this port can pull everything the cache holds. The port does not accept writes, whatever the credentials: a push or a delete is refused, in an air-gapped cluster as well, where the port serves the store itself rather than a cache in front of an upstream. The store accepts writes only on `4283/TCP`, which requires a client certificate.
`4283/TCP`	Writing: `d8 mirror push` through the Ingress, and the replica's own filling, replication and garbage collection.	Mutual TLS. The connection is refused during the TLS handshake unless the client presents a certificate issued by the module's certificate authority to the ingress controller or to the cache's own pod. After that, the write credentials are checked. `whitelistSourceRanges` is applied by the Ingress only: the port does not filter by source address. A connection made directly to the port, or to the `registry-push` Service inside the cluster, without such a certificate fails at TLS.
`4285/TCP`	Metrics.	`kube-rbac-proxy`: a request needs a token of a subject allowed to get the `prometheus-metrics` subresource of the `registry-storage` StatefulSet in `d8-system`. The registry's own metrics and debug listeners are behind it, on the loopback.

The ingress controller presents its certificate for any Ingress that names the `d8-system/registry-storage-ingress-client` secret, not only for the module's own. An ingress-nginx controller that lets an Ingress use a secret of another namespace (the `allow-cross-namespace-resources` option, enabled by default in controller versions before 1.12, 1.10 among them) therefore lets anyone who can create an Ingress and a Service in any namespace reach `4283/TCP` through them, outside `whitelistSourceRanges`. The write credentials are still checked on that path. To close it, turn `allow-cross-namespace-resources` off in the controller's configuration: the module's own Ingress is in `d8-system`, beside the secret, and keeps working.

A replica whose pod was created before `4281/TCP` started refusing writes still accepts a push on that port in an air-gapped cluster, until the pod is replaced. The `registry-storage` StatefulSet is updated with `OnDelete`: the module replaces the pods one at a time, each only while something else can serve the nodes in its place: another replica, the upstream, or, for the only replica of an air-gapped cache, its

new images already held on its master node (see How is the cache updated?). While a replacement waits, the module records an `UpdateBlocked` event on the `d8-system/registry-storage` StatefulSet.

Do not delete the pod to hurry this along. The new pod pulls its images from the store it is itself part of: deleted before they are on its node, the only replica of an air-gapped cache stays in `ImagePullBackOff`, and until it is back no node of the cluster can pull any image.

Air-gapped clusters

If no upstream is set in `primary.upstream`, the in-cluster cache is the only source of images, and images get into it only with `d8 mirror push`.

`d8 mirror push` writes through the publication endpoint: the `registry-push` Ingress in the `d8-system` namespace, at the `registry` domain built from publicDomainTemplate. The endpoint exists whenever the cache is enabled, `publicDomainTemplate` is set and the `cert-manager` module is enabled, including while an upstream is set, so that the images can be pushed before the upstream is removed. A push always lands on the leader; while there is no leader, it is refused.

This is the one path that can replace an image, so a push needs both credentials with write access — the account in the `registry-storage-push` Secret in `d8-system`, meant for `d8 mirror push` — and the client certificate the ingress presents to the storage: a leaked password alone is not enough. The certificate is also what tells the storage that the client address passed by the ingress can be trusted. To limit who can reach the endpoint, set `whitelistSourceRanges`. A push sent straight to a control-plane node, around the endpoint, is refused on either port of the cache. For what protects each port, including on a replica whose pod has not been replaced yet after an update, see What protects the ports on control-plane nodes.

If `publicDomainTemplate` is not set or the `cert-manager` module is disabled, an air-gapped cluster has no way to write into the store: `4281/TCP` refuses writes, and `4283/TCP` admits only the ingress controller and the cache itself. To bring images in, either set `publicDomainTemplate` and enable `cert-manager` (the endpoint appears on the next run of the module), or set `primary.upstream` to a registry the cluster can reach that holds the images, let the cache fill from it, and remove the upstream again.

Installing from a bundle

An air-gapped cluster can also be installed straight from a bundle made with `d8 mirror pull`. To do so, set `mode: Managed` in the `registry` ModuleConfig, enable `storage.cache` with its `storage.size`, describe the bundle in `storage.source`, set no `primary.upstream`, and pass the bundle to the installer with `--img-bundle-path`:

```
spec:
  enabled: true
  settings:
    mode: Managed
    storage:
      cache: true
      size: 50Gi
      source:
        bundleRef: d8-mirror-bundle
        expectedDigests: 459
```

A temporary registry on the first control-plane node serves the bundle until the in-cluster cache starts on that node, and the installer then hands the node over to the cache.

Installing from a bundle has the following limitations:

- In a cloud cluster, the `cloud-controller-manager` must be able to initialize the first control-plane node, which means the cloud API must be reachable from the cluster. Until the node loses the `node.cloudprovider.kubernetes.io/uninitialized` taint, the cache cannot be placed on it. The installer waits for that, and if it does not happen, it stops with an error naming the node and keeps the temporary registry, so that the node is not left with nothing to pull images from.
- A master NodeGroup with `systemType: DeckhouseEngine` does not support installing from a bundle: such a node runs none of the steps that set up the temporary registry. The installer refuses this combination before it creates any infrastructure. To install such a cluster without internet access, push the bundle with `d8 mirror push` into a registry inside your perimeter, and set `mode: Managed` with `primary.upstream` pointing at that registry.

Requirements for using the module

Before enabling the `registry` module to manage the pull paths for container images, make sure the cluster meets the following requirements:

- Nodes use containerd or containerd v2, set with the `defaultCRI` parameter in ClusterConfiguration.
- The cluster is fully managed by DP. The module does not work in Managed Kubernetes.
- The in-cluster cache uses disks on the control-plane nodes, in static and cloud clusters alike. In a cloud cluster that is the control-plane node's root disk unless another one is mounted at the cache directory (`/var/lib/deckhouse/registry` when the master NodeGroup has `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` otherwise). That disk must hold `storage.size` plus the reserve the store keeps free for the node, on top of what the node itself uses. Size `storage.size` for three releases of the image set together with the modules and the module catalog; see What happens when the cache's store is full?.

The v1 implementation

The v1 implementation of the module, used in DP up to and including 1.77, is described below. DP 1.78 no longer contains it; the description is kept because clusters are still running on it, and moving to the new configuration is work for the administrator that has to be done before the cluster is upgraded, not after.

✖ Prepare the cluster BEFORE upgrading to DP 1.78: the upgrade is blocked while the cluster uses the v1 implementation and the module is running in `Proxy` or `Local` mode, or in `Direct` mode without a configured `registry` ModuleConfig.

Installing DP 1.78 or later, or upgrading to it, does not create any objects of the v1 implementation. A new installation with `Proxy` or `Local` in the `deckhouse` ModuleConfig is refused by the installer: configure this module instead. The v1 implementation's modes are switched only by the code of DP 1.77 and earlier — so all preparation happens before upgrading to DP 1.78. Preparation has the following specifics:

- If the cluster uses the v1 implementation of the `registry` module running in `Proxy` mode, it must be switched to `Unmanaged` mode.
- If a cluster with the v1 implementation of the `registry` module is air-gapped (the module runs in `Local` mode), switching straight to `Unmanaged` mode does not work. Air-gapped clusters have a separate procedure, described in How do I migrate an air-gapped cluster from Local mode.
- If the cluster uses the v1 implementation of the `registry` module running in `Direct` mode, no mode switch is needed — only this module's configuration, written before the upgrade. Its objects are kept deliberately — DP 1.77 marks them to survive the removal of the v1 implementation — and they keep serving the in-cluster address nodes pull through, until this module's agent has taken that address over. The procedure is described in migrating from Direct mode.

In the v1 implementation of the module, working with the registry is configured not through the module's own settings, but through the `deckhouse` ModuleConfig, and it works as a state machine with four modes:

- `Direct` — direct access to an external registry through the fixed address `registry.d8-system.svc:5001/system/deckhouse`. The fixed address is what keeps Deckhouse images from being re-downloaded, and components from restarting, when registry parameters change.
- `Proxy` — an internal caching proxy registry on the control-plane nodes, reachable at that same fixed address; it reduces the number of requests to the external registry.
- `Local` — a full local copy of the registry inside the cluster, for air-gapped environments.
- `Unmanaged` — no in-cluster registry; the cluster reaches the external one directly. Comes in a configurable form, managed through the `deckhouse` ModuleConfig, and a deprecated non-configurable form, set at installation.

Of the four v1 modes listed above, two survive an upgrade to DP 1.78, for different reasons. `Unmanaged` is the mode whose objects served nothing to begin with, so removing them takes nothing away, and the handover happens on its own. `Direct` survives because the address its nodes pull images through is also served by the `registry` module: DP 1.77 marks the objects behind that address so that they outlive the removal of the v1 implementation, and they keep serving it until the agent takes it over — meaning such a cluster needs this module's configuration rather than a mode switch. Both paths are described in how does the migration to the `registry` module work. The two implementations never manage a cluster at the same time: whichever one is active decides what gets created at all, so there is no state where both configure the same node.

`Proxy` and `Local` modes leave nodes in a state that DP 1.78 and later cannot correctly account for or carry over. So the v1 implementation running in these modes is switched to `Unmanaged` first. That switch is only possible on DP 1.77 or earlier — that is, before upgrading to DP 1.78, which no longer

contains the v1 implementation.

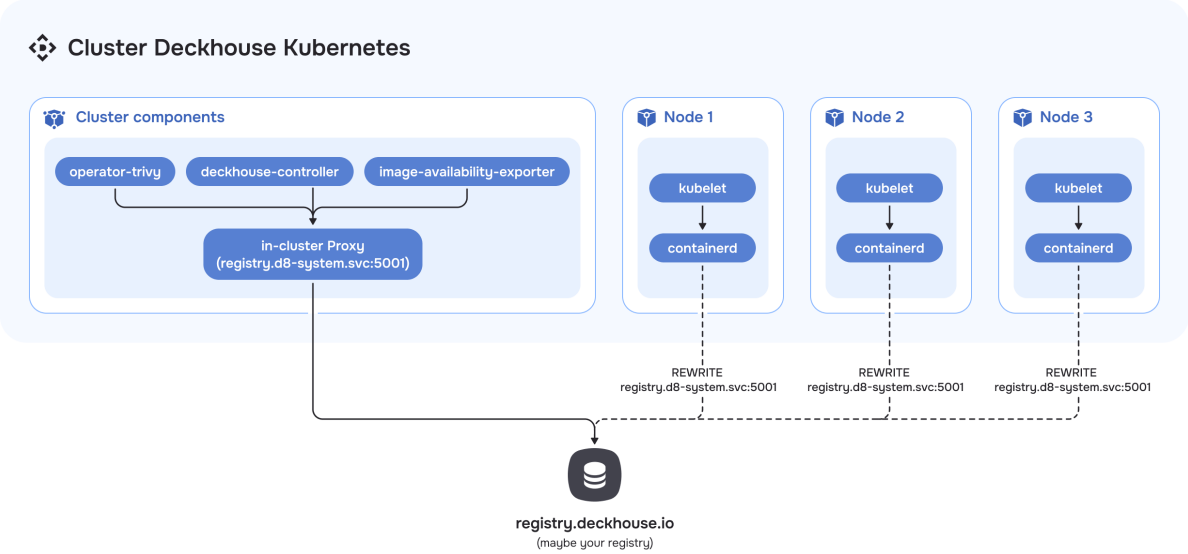
Mode switching restrictions

The restrictions on switching modes in the v1 implementation are as follows:

- Changing registry parameters and switching modes is only available after the bootstrap phase is fully complete.
- For the first switch, migrate the user's registry configurations first — the procedure is described in the FAQ.
- Switching to the non-configurable `Unmanaged` mode is only available from `Unmanaged`.

Direct mode architecture

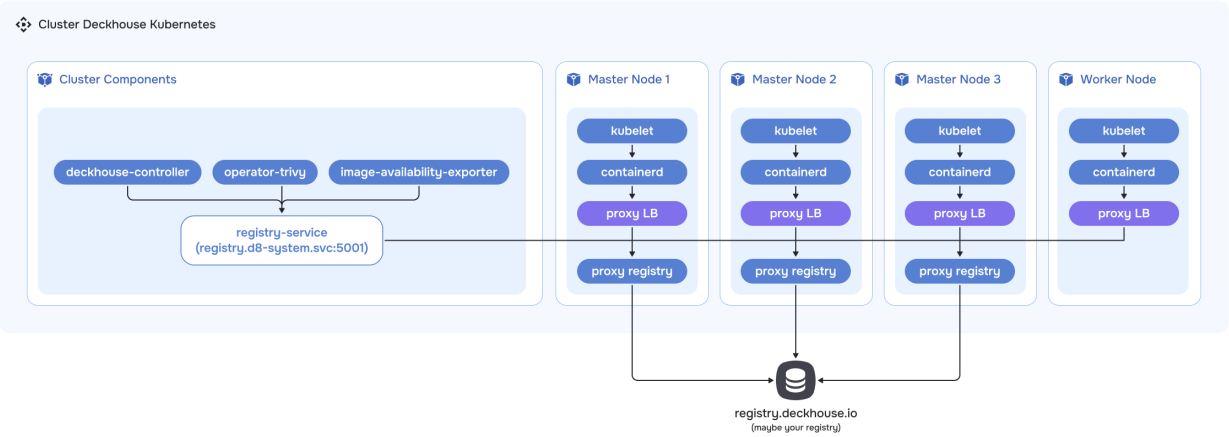
In `Direct` mode, requests to the registry are processed without intermediate caching. CRI requests are redirected based on the containerd configuration. Components that access the registry directly — `operator-trivy`, `image-availability-exporter`, `deckhouse-controller`, and others — go through the in-cluster proxy on the control-plane nodes.



Proxy mode architecture

> ⚠ It is recommended to use separate disks for storing registry data (`/opt/deckhouse/registry`) and etcd data. Using a single disk may lead to etcd performance degradation while the registry is operating.

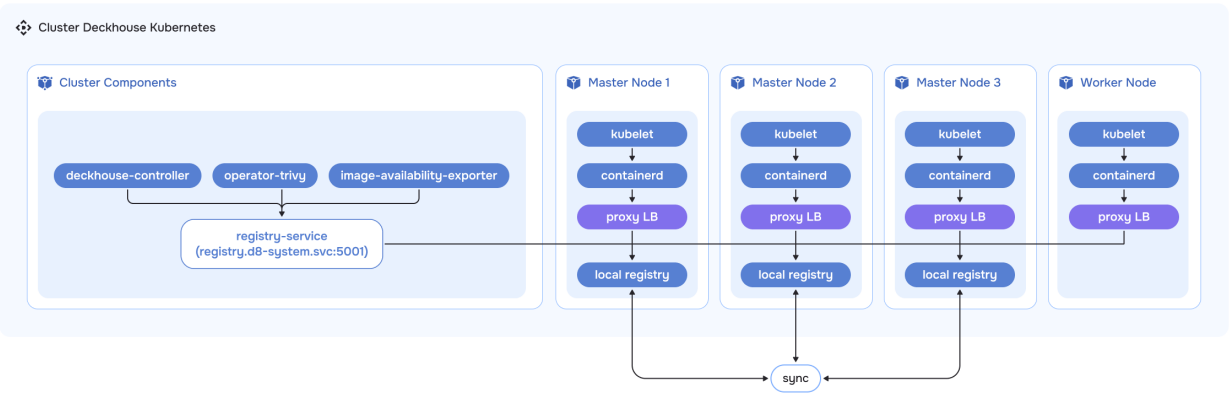
The caching proxy registry runs as static pods on the control-plane nodes and stores cached data in `/opt/deckhouse/registry`. A load balancer sits in front of it on each node, and the containerd configuration points at that load balancer. Components that access the registry directly go through the caching proxy registry.



Local mode architecture

⚠ It is recommended to use separate disks for storing registry data (`/opt/deckhouse/registry`) and etcd data. Using a single disk may lead to etcd performance degradation while the registry is operating.

`Local` mode keeps a full copy of the registry inside the cluster, synchronized between replicas on the control-plane nodes and populated with the `d8` tool using `d8 mirror push`/`d8 mirror pull`. Otherwise it works the same way as the caching proxy.



ℹ **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module registry: configuration

ℹ Included in extensions: **Advanced Infrastructure Security, Billing**

⚠ The module lifecycle stage: Preview

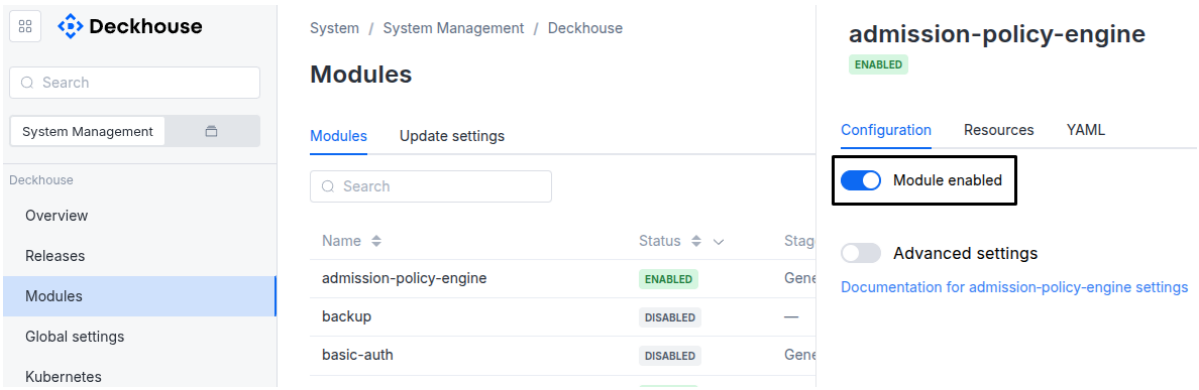
The module has 22 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable registry
```

- **Using ModuleConfig `registry`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `registry` (create it if necessary);

  Example of a manifest to enable module `registry`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: true
```

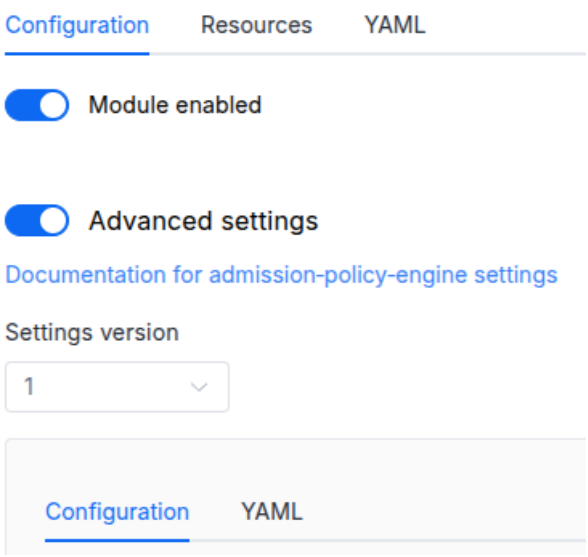
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `registry` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `registry`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `registry` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc registry
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `registry` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `registry`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: registry
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.https**
    object

    What certificate type to use.

    This parameter completely overrides the `global.modules.https` settings.

    Examples:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**
  object

  Parameters for certmanager.

  - **settings.https.certManager.clusterIssuerName**
    string

    What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

    Default: `letsencrypt`

    Example:

    ```
    clusterIssuerName: letsencrypt
    ```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the registry ingress.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager`: The registry ingress is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate`: The registry ingress is accessed over HTTPS using a certificate from the `d8-system` namespace.
  - `Disabled`: In this mode, the registry ingress can only be accessed over HTTP.
  - `OnlyInURI`: The registry ingress will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for the registry.

Optional. By default, the `modules.ingressClass` global value is used.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.mode**
  string

  Whether the module manages how the cluster pulls images.

  - `Unmanaged` — the module manages nothing. No components are created and no node configuration is written; the cluster keeps pulling from the registry it was installed with. This is the default, so enabling the module changes nothing on its own.

    Asking for `Unmanaged` on a cluster the module was managing does not take effect instantly: image references all over the cluster name the in-cluster registry, and they move to the upstream one only as each module's release is rendered again. So the module stops advertising its address at once and keeps serving it until nothing names it any more, which normally takes a few minutes. On a cluster that was installed with the module already managing, it also writes the registry it was fetching from back into the cluster's registry credentials — otherwise nothing would remember where images come from, since the installer had put the in-cluster registry there. Until then its components are still running, and `D8RegistryDrainStuck` reports a withdrawal that cannot finish.

  - `Managed` — the module owns the pull path: it configures the container runtime on every node through its node agent, and optionally runs an in-cluster cache.

  There is no choice of implementation. A cluster that has never run the v1 implementation of this module (DP up to and including 1.77) always uses the current one; a cluster that has runs the v1 implementation until it is brought to its `Unmanaged` state, after which the migration completes on its own.

  Default: `Unmanaged`

  Allowed values: `Managed`, `Unmanaged`

- **settings.primary**
  object

  The single authoritative source of Deckhouse component images. Applies only when `mode` is `Managed`.

  Additional registries are declared as separate RegistryUpstream resources rather than here: a module or a user bringing their own registry must not have to edit a ModuleConfig owned by someone else.

  - **settings.primary.upstream**
    object

    The registry to pull Deckhouse component images from.

    If omitted, the cluster is air-gapped: the in-cluster cache becomes authoritative and is populated with `d8 mirror push`. Omitting it requires `storage.cache: true` and `storage.source`.

    Example:

    ```
    upstream:
      host: registry.deckhouse.io
      path: "/deckhouse/ee"
      scheme: HTTPS
      auth:
        license: DECKHOUSE_LICENSE_KEY
    ```

    - **settings.primary.upstream.auth**
      object

      Credentials for the registry.

      - **settings.primary.upstream.auth.license**
        string

        The Deckhouse license key.

        A shorthand for the `username`/`password` pair used with `registry.deckhouse.io`. Mutually exclusive with them.

Changing the license changes the registry credentials, so it goes through the same preflight probe as an address change: the new credentials are verified before the cluster is switched over, and the last known good ones are kept if the probe fails.

- **settings.primary.upstream.auth.password**
  string

  The password for basic authentication.

- **settings.primary.upstream.auth.username**
  string

  The username for basic authentication.

- **settings.primary.upstream.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the registry when `scheme` is `HTTPS`.

  If not specified, the system trust store is used.

- **settings.primary.upstream.host**
  string

  Required value

  The registry host, optionally with a port.

  Without a protocol and repository prefix, they are specified in `scheme` and `path`, respectively.

  Pattern: `^[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?(\.[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?)*(:[0-9]{1,5})?$`

  Length: `1..255`

  Examples:

  ```
  host: registry.deckhouse.io
  ```

  ```
  host: my-private-registry.com:5000
  ```

- **settings.primary.upstream.mirrors**
  array of objects

  Additional addresses serving **the same** content as the primary registry, used for failover and load balancing.

  Mirrors are not separate sources: the cache holds one de-duplicated set regardless of which mirror the images came from. A registry with different content is an additional upstream and belongs in a RegistryUpstream resource.

  - **settings.primary.upstream.mirrors.auth**
    object

    Credentials for the registry.

    - **settings.primary.upstream.mirrors.auth.license**
      string

      The Deckhouse license key.

      A shorthand for the `username`/`password` pair used with `registry.deckhouse.io`. Mutually exclusive with them.

      Changing the license changes the registry credentials, so it goes through the same preflight probe as an address change: the new credentials are verified before the cluster is switched over, and the last known good ones are kept if the probe fails.

    - **settings.primary.upstream.mirrors.auth.password**
      string

      The password for basic authentication.

- **settings.primary.upstream.mirrors.auth.username**
  string

  The username for basic authentication.

- **settings.primary.upstream.mirrors.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror.

- **settings.primary.upstream.mirrors.host**
  string

  Required value

  The mirror host, optionally with a port.

  Without a protocol and repository prefix, they are specified in `scheme` and `path`, respectively.

  Pattern: `^[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?(\.[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?)*(:[0-9]{1,5})?$`

  Length: `1..255`

- **settings.primary.upstream.mirrors.path**
  string

  The repository prefix inside the mirror.

- **settings.primary.upstream.mirrors.scheme**
  string

  The protocol used to reach the mirror.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **settings.primary.upstream.path**
  string

  The repository prefix inside the registry.

  Example:

  ```
  path: "/deckhouse/ee"
  ```

- **settings.primary.upstream.scheme**
  string

  The protocol used to reach the registry.

  Use `HTTP` only for insecure, trusted registries.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **settings.storage**
  object

  The in-cluster registry cache. Applies only when `mode` is `Managed`.

  - **settings.storage.cache**
    boolean

    Whether pulls go through the in-cluster cache on the master nodes.

    This is one of the two configuration axes; the other is whether `primary.upstream` is set. Together they cover every supported layout:

    - `cache: false` with an upstream — nodes pull straight from the upstream, no cache is deployed.
    - `cache: true` with an upstream — a pass-through cache, filled from the upstream.
    - `cache: true` without an upstream — air-gap: the cache is the only source of images and is filled with `d8 mirror push`.

Turning the cache on or off is a safe, idempotent reconfiguration. Removing the upstream while the cache is on is the one transition with a condition: it takes effect only once the cache leader holds the whole expected image set, so nodes are never cut off.

Default: `false`

- **settings.storage.garbageCollection**
  object

  When the cache reclaims the disk taken by releases the cluster has moved past.

  Needed because nothing else ever removes anything: every release adds a slice of the repository, so a cluster that lives for years fills its store and then stops being able to pull.

  A collection does not interrupt serving: the replica keeps serving every image it holds. It does not start while the store is being filled or a `d8 mirror push` is in progress, and while it runs the replica stops filling its own store. A `d8 mirror push` started during a collection can fail and has to be retried. Only one replica collects at a time, so the others are unaffected.

  - **settings.storage.garbageCollection.enabled**
    boolean

    Whether the cache reclaims its disk on its own: on the schedule, once it holds a new image set, and when it refuses writes.

    Turning this off leaves the store to grow without bound, which only makes sense with a disk large enough that it never matters. A collection created by hand as a RegistryOperation object still runs.

    Default: `true`

  - **settings.storage.garbageCollection.schedule**
    string

    A five-field cron expression, in UTC.

    Defaults to a night hour. If the `master` node group has a maintenance window, the start of that window is used instead — that being a time the operator has already declared safe for disruption.

    Pattern: `^\s*\S+\s+\S+\s+\S+\s+\S+\s+\S+\s*$`

    Examples:

    ```
    schedule: 17 3 * * *
    ```

    ```
    schedule: 0 2 * * Sun
    ```

- **settings.storage.size**
  string

  How much the cache may hold on each master: the budget of its store, which the registry enforces on every write.

  Once the store holds this much, it refuses new writes — a fill, a replication, a `d8 mirror push` — and keeps serving everything it holds. It never deletes anything to make room; that is the garbage collection’s job.

  Independently of this value, the store never takes the space kept free on its filesystem: on a filesystem shared with the node, the kubelet’s soft eviction threshold (10% of the filesystem, at most 40 GiB) plus a margin (10%, at most 20 GiB); on a disk of the store’s own, the margin alone. So an unset or too large value cannot fill a master’s disk — but it leaves the reserve as the only limit, which the `D8RegistryStoreUnbounded` alert reports.

  The reserve is taken from the full size of the filesystem, while free space does not include the blocks the filesystem keeps for root. On ext4 these are 5% by default, so on a filesystem under 200 GiB the store and everything else on it get about 75% of the disk rather than 80%.

  Size it for three releases of the image set (the deployed one, the previous one kept for a rollback, and the next one) together with the modules the cluster runs and the catalog of modules the module sources offer. A replica copies the next release ahead of the

update only while the room it has left, in the budget and on the filesystem above the reserve, is at least a fifth of the budget (or of the filesystem, when `storage.size` is not set); with an upstream this applies to every replica. In an air-gapped cluster the next release arrives on the leader with `d8 mirror push` and needs that room, and the other replicas copy it from the leader under the same rule. The store holds the union of the releases' images: a layer they share is stored once. Make sure the value fits on the disk under the store above the reserve.

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Example:

```
size: 50Gi
```

- **settings.storage.source**
  object

  The image set the cache is expected to hold.

  Required in an air-gapped cluster: with no upstream to fall back on, completeness must be decidable before the cache can be trusted as the only source.

  - **settings.storage.source.bundleRef**
    string

    The name of the image set, for example the bundle pushed with `d8 mirror push`.

    Example:

    ```
    bundleRef: d8-mirror-bundle
    ```

  - **settings.storage.source.expectedDigests**
    integer

    The number of distinct digests the set contains.

    Allowed values: `0 <= X`

    Example:

    ```
    expectedDigests: 459
    ```

- **settings.whitelistSourceRanges**
  array of strings

  A list of CIDR-formatted addresses allowed to connect to the registry's publication endpoint, the Ingress that `d8 mirror push` writes through. If not specified, connections from any address are allowed.

  Applied by the Ingress only. The write port of the in-cluster cache behind it (`4283/TCP` on the control-plane nodes) does not filter by source address: it refuses any connection without a client certificate that the module's certificate authority issued to the ingress controller or to the cache's own pods. The cache's read port (`4281/TCP`) does not accept writes at all, so the write port is the only one a push can land on.

  The ingress controller presents its certificate for any Ingress that names the module's ingress client secret. If the controller lets an Ingress use a secret of another namespace (`allow-cross-namespace-resources`, enabled by default in ingress-nginx before 1.12), an Ingress of any namespace can reach the write port this way, outside this list; the write credentials are still checked. See What protects the ports on control-plane nodes.

  Example:

  ```
  whitelistSourceRanges:
  - 10.0.0.0/10
  - 192.168.0.0/16
  ```

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))?$`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The registry module: Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> ⚠ The module lifecycle stage: Preview

RegistryConfig

Short names: rcfg

Scope: Cluster
Version: v1alpha1

Defines the resolved registry configuration of the cluster.

In normal operation the object is rendered by the module from ModuleConfig/registry. It is a contract rather than Deckhouse property: an administrator may edit it directly when the Deckhouse operator is unavailable, and the change is applied on the fly by the registry controller and the storage syncer without a helm re-render.

Only one object named `registry` is used.

- **spec**
  object

  Required value

  - **spec.mode**
    string

    Whether the module manages the image pull path.

    - `Managed` — the module owns the pull path and creates its components.
    - `Unmanaged` — the module owns nothing; no components are created.

    Default: `Managed`

    Allowed values: `Managed`, `Unmanaged`

  - **spec.primary**
    object

    The single authoritative source of Deckhouse component images.

    Additional registries are declared as separate RegistryUpstream objects. More than one primary source is not supported: it would make both “is the cache complete” and “what fills the cache in an air-gapped cluster” ambiguous.

    - **spec.primary.upstream**
      object

      The registry to pull Deckhouse component images from.

      If omitted, the cluster is air-gapped: the in-cluster cache becomes authoritative and is populated out of band with `d8 mirror push`. Omitting it requires `storage.cache: true`.

      - **spec.primary.upstream.auth**
        object

        Credentials for the registry.

Sensitive. Credentials are stored inline in the object rather than referenced from a Secret, so that the node agent stays self-sufficient while the API server is unreachable. Restrict access to registry resources accordingly.

- **spec.primary.upstream.auth.auth**
  string

  Pre-encoded `base64("username:password")`.

  Takes precedence over `username` and `password` when set.

  If the caller is not allowed to read the `registryconfigs/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.primary.upstream.auth.password**
  string

  The password for basic authentication.

  If the caller is not allowed to read the `registryconfigs/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.primary.upstream.auth.username**
  string

  The username for basic authentication.

- **spec.primary.upstream.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the registry when `scheme` is `HTTPS`.

  If omitted, the system trust store is used.

- **spec.primary.upstream.host**
  string

  Required value

  The registry host, optionally with a port.

  Example:

  ```
  host: registry.deckhouse.io
  ```

- **spec.primary.upstream.mirrors**
  array of objects

  Additional addresses serving **the same** content as the primary registry, used for failover and load balancing.

  Mirrors are not separate sources. The storage caches one de-duplicated set regardless of which mirror the images were pulled from.

  - **spec.primary.upstream.mirrors.auth**
    object

    Credentials for the mirror.

    Sensitive. See `primary.upstream.auth`.

    - **spec.primary.upstream.mirrors.auth.auth**
      string

      Pre-encoded `base64("username:password")`.

      If the caller is not allowed to read the `registryconfigs/sensitive` subresource, the API returns `<omitted>` instead of the value.

    - **spec.primary.upstream.mirrors.auth.password**
      string

      The password for basic authentication.

      If the caller is not allowed to read the `registryconfigs/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.primary.upstream.mirrors.auth.username**
  string

  The username for basic authentication.

- **spec.primary.upstream.mirrors.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror.

- **spec.primary.upstream.mirrors.host**
  string

  Required value

  The mirror host, optionally with a port.

  Example:

  ```
  host: registry-mirror.example.com
  ```

- **spec.primary.upstream.mirrors.path**
  string

  The repository prefix inside the mirror.

- **spec.primary.upstream.mirrors.scheme**
  string

  The protocol used to reach the mirror.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.primary.upstream.path**
  string

  The repository prefix inside the registry.

  Example:

  ```
  path: "/deckhouse/ee"
  ```

- **spec.primary.upstream.scheme**
  string

  The protocol used to reach the registry.

  Use `HTTP` only for insecure, trusted registries.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.storage**
  object

  Configures the in-cluster registry cache.

  - **spec.storage.cache**
    boolean

    Whether pulls go through the in-cluster cache on the master nodes.

    This is the first of the two configuration axes. The second is whether `primary.upstream` is set. Together they replace the former mode state machine:

    - `cache: false` with an upstream — the agent forwards pulls straight to the upstream and no storage is needed.
    - `cache: true` with an upstream — a pass-through cache, filled from the upstream.
    - `cache: true` without an upstream — air-gap: the cache is authoritative and is filled with `d8 mirror push`.

    Default: `false`

  - **spec.storage.garbageCollection**

object

When the cache reclaims the disk taken by releases the cluster has moved past.

Nothing else ever removes anything: every release adds a slice of the repository, so a cluster that lives for years fills its store and then stops being able to pull.

A collection does not interrupt serving: the replica keeps serving every image it holds. It does not start while the store is being filled or a `d8 mirror push` is in progress, and while it runs the replica stops filling its own store. A `d8 mirror push` started during a collection can fail and has to be retried. Only one replica collects at a time, so the others are unaffected.

- **spec.storage.garbageCollection.enabled**
  boolean

  Whether the cache reclaims its disk on its own: on the schedule, after a new image set, and when the store refuses writes. A RegistryOperation created by hand runs either way.

- **spec.storage.garbageCollection.schedule**
  string

  A five-field cron expression, in UTC.

  Defaults to a night hour, or to the start of the `master` node group’s maintenance window when one is set.

- **spec.storage.size**
  string

  The size of the persistent volume backing the cache.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  size: 50Gi
  ```

- **spec.storage.source**
  object

  The image set the cache is expected to hold.

  Required in an air-gapped cluster: with no upstream to fall back on, completeness must be decidable before it is safe to serve from the cache alone.

  - **spec.storage.source.bundleRef**
    string

    The name of the image set, for example the bundle pushed with `d8 mirror push`.

    Example:

    ```
    bundleRef: d8-mirror-bundle
    ```

  - **spec.storage.source.expectedDigests**
    integer

    The number of distinct digests the set contains.

    Allowed values: `0 <= X`

    Example:

    ```
    expectedDigests: 459
    ```

- **status**
  object
  - **status.conditions**
    array of objects

    The health summary written by the registry controller.

    - `Valid` — the resolved configuration is coherent.

- `UpstreamValid` — the result of the preflight probe of the primary upstream: reachable, credentials accepted, sentinel content in place. `False` means the controller is holding the last known good upstream and has **not** switched over, so a broken license or a wrong address cannot take the cluster down.
- `StorageConverged` — the storage has reached the state the configuration asks for.
- **status.conditions.lastTransitionTime**
  string

  Required value

  When the condition last changed.

- **status.conditions.message**
  string

  A human-readable explanation.

- **status.conditions.observedGeneration**
  integer

  The `spec` generation the condition was set for.

- **status.conditions.reason**
  string

  Required value

  A machine-readable reason for the condition.

- **status.conditions.status**
  string

  Required value

  The condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  The condition type.

- **status.effectiveUpstream**
  object

  The upstream actually in effect, which is not necessarily the configured one.

  This is the controller’s own record of the last upstream that passed its preflight probe, and it is what makes a bad change survivable: entering credentials that do not work, or switching to a license without access, leaves this field — and the cluster — on the previous working upstream, while the `UpstreamValid` condition reports why.

  It also holds the upstream through the air-gap transition, where the configuration already says “none” but the cache is not complete yet.

  Empty means no upstream is in effect: either the cluster is air-gapped, or nothing has been configured yet.

  - **status.effectiveUpstream.ca**
    string

    A PEM-encoded certificate authority bundle used to verify the registry.

  - **status.effectiveUpstream.host**
    string

    The registry host, optionally with a port.

  - **status.effectiveUpstream.mirrors**
    array of objects

    Additional addresses serving the same content.

    - **status.effectiveUpstream.mirrors.ca**
      string

A PEM-encoded certificate authority bundle used to verify the mirror.

- **status.effectiveUpstream.mirrors.host**
  string

  Required value

  The mirror host, optionally with a port.

- **status.effectiveUpstream.mirrors.path**
  string

  The repository prefix inside the mirror.

- **status.effectiveUpstream.mirrors.scheme**
  string

  The protocol used to reach the mirror.

  Allowed values: `HTTP`, `HTTPS`

- **status.effectiveUpstream.path**
  string

  The repository prefix inside the registry.

- **status.effectiveUpstream.scheme**
  string

  The protocol used to reach the registry.

  Allowed values: `HTTP`, `HTTPS`

- **status.effectiveUpstreamAuthDigest**
  string

  Identifies the credentials in effect without disclosing them.

  Needed because “has the configuration changed?” has to include the credentials: a rotated license under an unchanged address is exactly the change that must be probed before the cluster is switched onto it. With the credentials no longer recorded here, comparing this against a digest of the configured ones is what keeps that question answerable.

- **status.observedGeneration**
  integer

  The `spec` generation the controller last acted on.

RegistryMirror

Scope: Cluster
Version: v1alpha1

Declares a registry near a group of nodes that holds a copy of the primary image set, for those nodes to pull from instead of the in-cluster storage or the upstream.

Meant for nodes far from the control plane, such as workers in another region that pull large images across a slow or expensive link. The administrator runs and fills the mirror: a Harbor proxy-cache or replication project, a registry filled with `d8 mirror push`, or anything else that serves the image set by digest. The cluster only decides which nodes ask the mirror, and for what.

The registry agent on a selected node asks the mirror for manifests and blobs named by digest, and for nothing else. Tags are resolved where they are resolved without a mirror, so which version runs stays the cluster’s decision. Whatever the mirror answers short of the content — a miss, a refusal, a server error, no answer at all — sends the request on to the in-cluster storage or the upstream, as without a mirror. Content that does not match its digest is not relayed, and the agent stops asking that address of the mirror for that digest for an hour.

A node is served by one mirror at most. Objects are accepted in creation order, starting with the earliest created object. An object that selects a node already served by an earlier one is not accepted at all, on any of the nodes it selects, and the message of its `Accepted` condition names the node and the object that holds it.

Applied only in the `Managed` mode of the module, where the registry agent serves the pull path of the nodes. In the other modes the controller that writes `status` does not run, so `status` keeps what it last said, including `Accepted=True` and the number of nodes, while no node applies the mirror.

- **spec**
  object

  Required value

  - **spec.nodeSelector**
    object

    Required value

    The nodes that ask the mirror first, selected by their labels.

    An empty selector selects no node. A selector that selects no node is accepted all the same, with `status.matchedNodes` at zero: a node group scaled to zero gets the mirror once its nodes appear.

    Example:

    ```
    nodeSelector:
      matchLabels:
        node.deckhouse.io/group: gpu-eu
    ```

    - **spec.nodeSelector.matchExpressions**
      array of objects

      Label requirements a node must meet.

      - **spec.nodeSelector.matchExpressions.key**
        string

        Required value

        The label name.

      - **spec.nodeSelector.matchExpressions.operator**
        string

        Required value

        The comparison method with `values`.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.nodeSelector.matchExpressions.values**
        array of strings

        The label values. Required with `In` and `NotIn`, empty with `Exists` and `DoesNotExist`.

    - **spec.nodeSelector.matchLabels**
      object

      Labels and their values. A node is selected when it has every listed label with the given value.

  - **spec.source**
    object

    Required value

    The mirror: where it is, under which prefix it serves the image set, and how to authenticate to it.

    The prefix replaces the image set's own: an image the cluster names `registry.d8-system.svc:5001/system/deckhouse/<repository>@<digest>` is asked of the mirror as `<host>/<path>/<repository>@<digest>`.

    - **spec.source.auth**
      object

      Credentials for the mirror. Read-only access is enough.

      Without them the mirror is asked anonymously. A mirror that asks for a token, as a public Harbor project does, gets an anonymous one. The credentials a pod's `imagePullSecrets` hold are never sent to the mirror.

      The controller transfers sensitive data to the module's secret, and the nodes for which

the mirror is configured read it from this secret.

- **spec.source.auth.auth**
  string

  The encoded value is `"username:password"` in Base64.

  Takes precedence over `username` and `password` when set. When both forms are set, they must name the same credentials.

  If the caller is not allowed to read the `registrymirrors/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Maximum length: `4096`

- **spec.source.auth.password**
  string

  The password for basic authentication.

  If the caller is not allowed to read the `registrymirrors/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Maximum length: `4096`

- **spec.source.auth.username**
  string

  The username for basic authentication.

  Maximum length: `1024`

- **spec.source.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror when `scheme` is `HTTPS`.

  Only `CERTIFICATE` blocks are accepted, and each must hold a valid X.509 certificate. A block of another type, such as a private key, or a certificate that cannot be read leaves the mirror rejected: `Accepted=False` with the reason `InvalidSpec`.

  Maximum length: `65536`

- **spec.source.host**
  string

  Required value

  Mirror address. If necessary, it is specified with a port.

  An IPv6 address is written in brackets, followed by a port.

  Length: `1..255`

  Example:

  ```
  host: harbor.eu.example.com
  ```

- **spec.source.mirrors**
  array of objects

  Other addresses that return the same content as the mirror. They are queried in order after the main mirror.

  - **spec.source.mirrors.auth**
    object

    Credentials for the address.

    The controller transfers sensitive data to the module’s secret, and the nodes for which the mirror is configured read it from this secret.

    - **spec.source.mirrors.auth.auth**
      string

      The encoded value is `"username:password"` in Base64.

If the caller is not allowed to read the `registrymirrors/sensitive` subresource, the API returns `<omitted>` instead of the value.

Maximum length: `4096`

- **spec.source.mirrors.auth.password**
  string

  The password for basic authentication.

  If the caller is not allowed to read the `registrymirrors/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Maximum length: `4096`

- **spec.source.mirrors.auth.username**
  string

  The username for basic authentication.

  Maximum length: `1024`

- **spec.source.mirrors.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the address.

  Maximum length: `65536`

- **spec.source.mirrors.host**
  string

  Required value

  Mirror address. If necessary, it is specified with a port.

  An IPv6 address is written in brackets, followed by a port.

  Length: `1..255`

- **spec.source.mirrors.path**
  string

  Required value

  The prefix the image set is served under there. Required, as `source.path` is.

  Length: `1..1024`

- **spec.source.mirrors.scheme**
  string

  The protocol used to reach the address.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.source.path**
  string

  Required value

  The prefix the mirror serves the image set under. Required: the image of an embedded module names the image set’s prefix as its whole repository, so a mirror serving the set at its root could not be asked for it.

  For a Harbor proxy-cache project of `registry.deckhouse.io`, it is the project followed by the path of the edition, such as `/<project>/deckhouse/ee`. For a project filled with `d8 mirror push <bundle> harbor.example.com/<project>/<path>`, it is `/<project>/<path>`.

  Length: `1..1024`

  Example:

```
path: "/d8-proxy/deckhouse/ee"
```

        - **spec.source.scheme**
          string

          The protocol used to reach the mirror.

          Default: `HTTPS`

          Allowed values: `HTTP`, `HTTPS`

- **status**
  object
  - **status.conditions**
    array of objects

    The arbitration result with a machine-readable reason.

    - `Accepted` — whether the mirror is laid out for the nodes it selects. The reason `NodeConflict` means a node it selects is served by an earlier object; `InvalidSpec` means a field of `spec` is invalid.
    - **status.conditions.lastTransitionTime**
      string

      Required value

      The time of the last change to the condition.

    - **status.conditions.message**
      string

      A human-readable explanation.

    - **status.conditions.observedGeneration**
      integer

      The `spec` generation the condition was set for.

    - **status.conditions.reason**
      string

      Required value

      A machine-readable reason for the condition.

    - **status.conditions.status**
      string

      Required value

      The condition status.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      The condition type.

  - **status.matchedNodes**
    integer

    The number of nodes for which the mirror is configured. The value is zero until the object is accepted.

  - **status.observedGeneration**
    integer

    The `spec` generation the controller last acted on.

RegistryNode

Short names: rnode

Scope: Cluster
Version: v1alpha1

Defines the complete registry layout of a single node, as compiled for its agent.

One object per node, named after the node, with an `ownerReference` on it. Written by the registry controller; `status` is written by the agent only.

The object is **flat** and, apart from one deliberate reference, self-contained: certificate authority bundles and routes are inlined, and the agent keeps a full copy on disk so it keeps serving images while the API server is unreachable — at which point there is nothing left to dereference. The exception is credentials: they are named rather than held (`auth.secretRef` points at the Secret this module resolves them into), which is what lets every node read this object without also reading every registry credential in the cluster. For the same reason the controller compiles every accepted RegistryUpstream into `spec.additionalRoutes` instead of having the agent watch N objects: consistency across many objects cannot be guaranteed when the API is down.

The object carries **no per-node secrets**. Everything in it is global configuration, identical on every node. This is what makes it safe for the whole `system:nodes` group to read, which is how the agent gets access using the kubelet's kubeconfig.

- **spec**
  object

  Required value

  - **spec.additionalRoutes**
    array of objects

    The accepted RegistryUpstream objects, compiled by the controller.

    Always transit, never cached: the agent proxies the pull and streams it through without writing to disk.

    - **spec.additionalRoutes.auth**
      object

      Where the credentials for this endpoint are kept.

      Named rather than carried. This resource is cluster-scoped and the node agent's role is bound to the `system:nodes` group, so a credential written here would be readable through the API by every kubelet in the cluster — including credentials for registries that node never pulls from. The component that reads this resolves the reference once and keeps the result, so it still works when the API server does not.

      - **spec.additionalRoutes.auth.secretRef**
        object

        The Secret holding the credentials, and the key inside it.

        - **spec.additionalRoutes.auth.secretRef.key**
          string

          Required value

          Key inside it. One Secret holds every credential the module resolved, keyed by what it belongs to.

        - **spec.additionalRoutes.auth.secretRef.name**
          string

          Required value

          Name of the Secret in the `d8-system` namespace.

    - **spec.additionalRoutes.ca**
      string

      A PEM-encoded certificate authority bundle used to verify the upstream.

    - **spec.additionalRoutes.host**
      string

      Required value

      The upstream host, optionally with a port.

      Example:

      ```
      host: registry-vendor.example.com
      ```

    - **spec.additionalRoutes.match**

string

Required value

The registry host intercepted on this node.

Example:

```
match: images.virtualization.example.com
```

- **spec.additionalRoutes.mirrors**
  array of objects

  Additional addresses serving the same content.

  - **spec.additionalRoutes.mirrors.auth**
    object

    Where the credentials for this mirror are kept.

    Named rather than carried, for the same reason as the endpoint above: this resource is readable by every node in the cluster. A mirror can need credentials of its own, so it can name a key of its own.

    - **spec.additionalRoutes.mirrors.auth.secretRef**
      object

      The Secret holding the credentials, and the key inside it.

      - **spec.additionalRoutes.mirrors.auth.secretRef.key**
        string

        Required value

        Key inside it.

      - **spec.additionalRoutes.mirrors.auth.secretRef.name**
        string

        Required value

        Name of the Secret in the `d8-system` namespace.

  - **spec.additionalRoutes.mirrors.ca**
    string

    A PEM-encoded certificate authority bundle used to verify the mirror.

  - **spec.additionalRoutes.mirrors.host**
    string

    Required value

    The mirror host, optionally with a port.

  - **spec.additionalRoutes.mirrors.path**
    string

    The repository prefix inside the mirror.

  - **spec.additionalRoutes.mirrors.scheme**
    string

    The protocol used to reach the mirror.

    Default: `HTTPS`

    Allowed values: `HTTP`, `HTTPS`

- **spec.additionalRoutes.path**
  string

  The repository prefix inside the upstream.

- **spec.additionalRoutes.scheme**
  string

  The protocol used to reach the upstream.

Default: `HTTPS`

Allowed values: `HTTP`, `HTTPS`

- **spec.backends**
  array of objects

  The sources of the **primary** image set, in priority order.

  With the cache on, this is the storage followed by the upstream as a fallback while the cache is still filling; the upstream entry disappears from the layout when the cluster goes air-gap. With the cache off, it is the upstream alone.

  - **spec.backends.auth**
    object

    Where the credentials for this endpoint are kept.

    Named rather than carried. This resource is cluster-scoped and the node agent's role is bound to the `system:nodes` group, so a credential written here would be readable through the API by every kubelet in the cluster — including credentials for registries that node never pulls from. The component that reads this resolves the reference once and keeps the result, so it still works when the API server does not.

    - **spec.backends.auth.secretRef**
      object

      The Secret holding the credentials, and the key inside it.

      - **spec.backends.auth.secretRef.key**
        string

        Required value

        Key inside it. One Secret holds every credential the module resolved, keyed by what it belongs to.

      - **spec.backends.auth.secretRef.name**
        string

        Required value

        Name of the Secret in the `d8-system` namespace.

  - **spec.backends.ca**
    string

    A PEM-encoded certificate authority bundle used to verify the backend.

  - **spec.backends.host**
    string

    Required value

    The backend host, optionally with a port.

    Example:

    ```
    host: registry.d8-system.svc:5001
    ```

  - **spec.backends.mirrors**
    array of objects

    Other addresses of the same backend. The agent recognises requests that name them, but pulls the primary image set only from the endpoint.

    - **spec.backends.mirrors.auth**
      object

      Where the credentials for this mirror are kept.

      Named rather than carried, for the same reason as the endpoint above: this resource is readable by every node in the cluster. A mirror can need credentials of its own, so it can name a key of its own.

      - **spec.backends.mirrors.auth.secretRef**
        object

The Secret holding the credentials, and the key inside it.

- **spec.backends.mirrors.auth.secretRef.key**
  string

  Required value

  Key inside it.

- **spec.backends.mirrors.auth.secretRef.name**
  string

  Required value

  Name of the Secret in the `d8-system` namespace.

- **spec.backends.mirrors.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror.

- **spec.backends.mirrors.host**
  string

  Required value

  The mirror host, optionally with a port.

- **spec.backends.mirrors.path**
  string

  The repository prefix inside the mirror.

- **spec.backends.mirrors.scheme**
  string

  The protocol used to reach the mirror.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.backends.name**
  string

  Required value

  The backend’s role.

  - `Storage` — the in-cluster cache.
  - `Upstream` — the primary upstream, used either as the only backend or as a fallback while the cache fills.

  Allowed values: `Storage`, `Upstream`

- **spec.backends.path**
  string

  The repository prefix inside the backend.

- **spec.backends.scheme**
  string

  The protocol used to reach the backend.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.cache**
  boolean

  The cache axis for this node.

  `true` sends the agent to the in-cluster storage, `false` straight to the primary upstream.

- **spec.primaryMirror**
  object

  The accepted RegistryMirror selecting this node, if any: a nearby copy of the primary image

set.

The agent asks it before `backends`, for manifests and blobs named by digest only. Whatever it answers short of the content — a miss, a refusal, a failure — sends the request on to `backends`, and content that does not match its digest is not relayed.

- **spec.primaryMirror.auth**
  object

  The location of the access keys to the mirror.

  Named rather than carried, for the same reason as in `backends`: this resource is readable by every node in the cluster.

  - **spec.primaryMirror.auth.secretRef**
    object

    The Secret holding the credentials, and the key inside it.

    - **spec.primaryMirror.auth.secretRef.key**
      string

      Required value

      Key.

    - **spec.primaryMirror.auth.secretRef.name**
      string

      Required value

      Name of the Secret in the `d8-system` namespace.

- **spec.primaryMirror.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror.

- **spec.primaryMirror.host**
  string

  Required value

  The mirror host, optionally with a port.

  Example:

  ```
  host: harbor.eu.example.com
  ```

- **spec.primaryMirror.mirrors**
  array of objects

  Other addresses serving the same content, asked in order after `host`.

  - **spec.primaryMirror.mirrors.auth**
    object

    The location of the access keys to the address.

    - **spec.primaryMirror.mirrors.auth.secretRef**
      object

      The Secret holding the credentials, and the key inside it.

      - **spec.primaryMirror.mirrors.auth.secretRef.key**
        string

        Required value

        Key.

      - **spec.primaryMirror.mirrors.auth.secretRef.name**
        string

        Required value

        Name of the Secret in the `d8-system` namespace.

        - **spec.primaryMirror.mirrors.ca**
          string

          A PEM-encoded certificate authority bundle used to verify the address.

        - **spec.primaryMirror.mirrors.host**
          string

          Required value

          The host. If necessary, it is specified with a port.

        - **spec.primaryMirror.mirrors.path**
          string

          The prefix the image set is served under there.

        - **spec.primaryMirror.mirrors.scheme**
          string

          The protocol used to reach the address.

          Default: `HTTPS`

          Allowed values: `HTTP`, `HTTPS`

    - **spec.primaryMirror.name**
      string

      Required value

      The RegistryMirror this comes from.

      Example:

      ```
      name: gpu-eu
      ```

    - **spec.primaryMirror.path**
      string

      The prefix the mirror serves the image set under, in place of the image set’s own.

      Example:

      ```
      path: "/d8-proxy/deckhouse/ee"
      ```

    - **spec.primaryMirror.scheme**
      string

      The protocol used to reach the mirror.

      Default: `HTTPS`

      Allowed values: `HTTP`, `HTTPS`

- **status**
  object

  Written by the agent only, reporting containerd-side facts.

  It carries no storage completeness: that is a property of the storage and lives in RegistryStorage `.status.replicas`. Any node can patch any `RegistryNode` status, because RBAC cannot scope `list`/`watch` by field selector, so nothing here may be load-bearing for the pull path — it is observability.

    - **status.activeBackends**
      array of strings

      The backends the agent is currently able to use.

    - **status.conditions**
      array of objects

      The detailed state reported by the agent.

        - `Reconciled` — whether the spec has been applied. The reason distinguishes a normal reconcile from one applied out of the on-disk copy (`AppliedFromCache`): the agent

could not get a usable spec from the API server. Most often the credentials it references could not be read; a read of the spec that failed while the status write after it went through, as on a short API server outage, looks the same. The agent reports the fallback by writing this status, so an agent that cannot write it at all, such as one on a node cut off from the API server, leaves the previous condition in place.

- **status.conditions.lastTransitionTime**
  string

  Required value

  When the condition last changed.

- **status.conditions.message**
  string

  A human-readable explanation.

- **status.conditions.observedGeneration**
  integer

  The `spec` generation the condition was set for.

- **status.conditions.reason**
  string

  Required value

  A machine-readable reason for the condition.

- **status.conditions.status**
  string

  Required value

  The condition status.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  The condition type.

- **status.observedGeneration**
  integer

  The `spec` generation the agent last applied.

- **status.proxyListening**
  boolean

  Whether the agent is serving the registry endpoint.

- **status.reconciled**
  boolean

  Whether the agent has applied the spec it currently sees.

- **status.staleStorageDataBytes**
  integer

  How much of the in-cluster cache’s data is still on this node while no cache is configured.

  The blobs are left behind deliberately when the cache is turned off: turning it back on then refills from what is already there rather than from scratch, and over a slow link that difference is hours. What would not be acceptable is keeping them silently, since nothing else in the module would ever mention the disk they occupy — so the agent measures them and an alert says so.

  To reclaim the space, remove the contents of the cache directory on the node, keeping the directory itself, which the agent has mounted: `/var/lib/deckhouse/registry` on a node of a NodeGroup with `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` on other nodes. Absent while a cache is configured, and on a node that never held one.

RegistryOperation

Short names: rop

Scope: Cluster
Version: v1alpha1

An operation every replica of the registry cache performs on its own store, one replica at a time.

Images keep being served throughout.

There are three operation types:

- `GarbageCollection` removes from the cache what the releases the cluster has moved past left there.
- `Verify` reads blobs of the store and compares each with its digest: every blob, or those of the images named in `spec.verify.digests`. What does not match is recorded in `RegistryStorage.status.replicas[].corruptedBlobs`. Besides, every replica checks its whole store in the background about once an hour, so a `Verify` is for starting a check manually.
- `Repair` checks the images named in `spec.repair.digests` and replaces their broken or missing blobs with whole copies from another replica of the cache or, when the cluster has one, from the upstream. A blob is downloaded beside the store, checked against its digest and only then put in place, so no image is removed and nothing is lost if no whole copy is found. The module creates a `Repair` itself for the broken blobs a replica finds, with the cause `CorruptionDetected`.

`Verify` and `Repair` read from the disk at no more than 32 MiB/s, so a store of 15 GB takes a replica about eight minutes to check. Neither waits for a fill or a `d8 mirror push` to finish.

The module creates garbage collections itself:

- on the schedule set in `storage.garbageCollection` of the module configuration;
- once the cache holds a new image set — after a Deckhouse Platform update, or after a module is enabled or updated;
- when the store refuses writes and so can never take in a new set — once per deployed Deckhouse Platform release.

With `storage.garbageCollection.enabled: false` the module creates no operations. An operation created by hand runs either way.

A garbage collection starts on a replica only when its store is not being written to: while the cache is being filled or an image set is being pushed with `d8 mirror push`, the replica waits. A store that refuses writes — over its budget, or down to the space kept free on its filesystem — is not waited for: a fill cannot go on until something frees room. A replica that has not picked the operation up, or is still held back by its store, an hour after the operation was accepted fails with the reason `DeadlineExceeded`. Once its turn comes, a replica looks at its store again: if a fill or a push has started meanwhile, it gives the turn back and waits for the store to go quiet.

Only one operation runs at a time. An operation created while an older one is not finished fails with the reason `AnotherOperationActive`.

A replica that is gone fails with the reason `ReplicaLost`:

- a replica collecting garbage, when it has not held the Lease `registry-storage-gc` in the `d8-system` namespace for five minutes;
- a replica waiting, when its node has not been `Ready` for ten minutes.

To cancel an operation, set the label `registry.deckhouse.io/cancel: "true"` on it. A replica that has not started does not start, and a replica that is running stops. What it has already removed stays removed. Deleting the RegistryOperation object cancels an operation too, but does not stop a replica whose operation has already started.

A finished operation is deleted a day after it finished.

Examples:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: gc-manual-
spec:
  type: GarbageCollection
  cause: Free space before pushing a new release.
```

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: verify-
spec:
  type: Verify
  cause: Check the cache after a disk failure on master-1.
  verify:
    nodes: ["master-1"]
```

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: repair-
spec:
  type: Repair
  repair:
    digests:
["sha256:0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef"]
```

- **spec**
  object

  Required value

  Does not change once the operation is created.

  - **spec.cause**
    string

    Why the operation was created, for a person reading it.

    The module sets it on the operations it creates itself.

    Maximum length: `1024`

  - **spec.repair**
    object

    The `Repair` operation.

    - **spec.repair.digests**
      array of strings

      Required value

      Images, by the digest of their manifest, or single blobs, by their digest. An image is repaired together with every blob it consists of.

      The image digests are listed by `d8 k get registryimages`; the broken blobs, by `RegistryStorage.status.replicas[].corruptedBlobs`.

      - **Element of the array**
        string

        Pattern: `^sha256:[a-f0-9]{64}$`

    - **spec.repair.nodes**
      array of strings

      The replicas where the operation runs, specified by node name. Empty means every replica. An operation that names a node without a replica of the cache fails with the reason `UnknownNode`.

      - **Element of the array**
        string

        Maximum length: `253`

  - **spec.type**
    string

Required value

What the operation does.

- `GarbageCollection` — remove from the cache what the releases the cluster has moved past left there. The deployed release, the previous one, anything newer and every tag that is not a version are kept.
- `Verify` — read blobs of the store and compare each with its digest. What does not match is recorded and repaired by a `Repair` the module creates.
- `Repair` — check the images named in `spec.repair` and replace their broken or missing blobs with whole copies.

Allowed values: `GarbageCollection`, `Verify`, `Repair`

- **spec.verify**
  object

  The `Verify` check. Absent means every blob of every replica.

  - **spec.verify.digests**
    array of strings

    Images, by the digest of their manifest, or single blobs, by their digest. An image is checked together with every blob it consists of. Empty means every blob of the store.

    The image digests are listed by `d8 k get registryimages`.

    - **Element of the array**
      string

      Pattern: `^sha256:[a-f0-9]{64}$`

  - **spec.verify.nodes**
    array of strings

    The nodes where the operation runs. Empty means every node. An operation that names a node without a replica of the cache fails with the reason `UnknownNode`.

    - **Element of the array**
      string

      Maximum length: `253`

- **status**
  object
  - **status.acceptedAt**
    string

    When the operation was accepted.

  - **status.deadline**
    string

    When a replica held back by a store that is being written to gives up.

    An hour after `acceptedAt`.

  - **status.finishedAt**
    string

    When the operation reached a terminal phase.

    The operation is deleted a day later.

  - **status.garbageCollection**
    object

    What a `GarbageCollection` removed, summed over the replicas.

    - **status.garbageCollection.reclaimedBytes**
      integer

      How many bytes the store shrank by: its blobs and uploads in progress, measured before the collection and after.

      Absent when the store could not be measured.

    - **status.garbageCollection.removedBlobs**

integer

How many blobs the sweep removed, as the registry counts them.

A sweep also removes what an earlier one failed to, so these blobs need not belong to the images deleted in this run. Absent when the sweep did not run.

- **status.garbageCollection.removedManifests**
  integer

  How many images were deleted by digest — the images the fill writes with no tag, where nearly all of a store's weight is.

- **status.garbageCollection.removedVersions**
  array of strings

  The Deckhouse Platform releases removed.

- **status.message**
  string

  A human-readable explanation of the reason.

- **status.phase**
  string

  Where the operation is.

  - `Pending` — no replica has started yet.
  - `Running` — at least one replica has started, and not every one has finished.
  - `Succeeded` — every replica finished the operation.
  - `Failed` — the operation was refused, or a replica did not finish it.
  - `Canceled` — the operation was canceled before every replica finished.

  Allowed values: `Pending`, `Running`, `Succeeded`, `Failed`, `Canceled`

- **status.reason**
  string

  A machine-readable reason for the phase.

  - `Pending`: `Accepted`.
  - `Running`: `InProgress`.
  - `Succeeded`: `Completed`.
  - `Failed`: `NoCache` (the cache is turned off), `NoReplicas` (no replica of the storage has reported yet), `AnotherOperationActive`, `UnknownNode` (`nodes` names a node without a replica), `ReplicaFailed`.
  - `Canceled`: `Canceled`.

- **status.replicas**
  array of objects

  One entry per replica the operation was handed to.

  Every replica of the storage that had reported when the operation was accepted is included.

  - **status.replicas.finishedAt**
    string

    When the replica finished.

  - **status.replicas.garbageCollection**
    object

    What a `GarbageCollection` removed from the replica's store.

    - **status.replicas.garbageCollection.reclaimedBytes**
      integer

      How many bytes the store shrank by: its blobs and uploads in progress, measured before the collection and after.

      Absent when the store could not be measured.

    - **status.replicas.garbageCollection.removedBlobs**
      integer

      How many blobs the sweep removed, as the registry counts them.

A sweep also removes what an earlier one failed to, so these blobs need not belong to the images deleted in this run. Absent when the sweep did not run.

- **status.replicas.garbageCollection.removedManifests**
  integer

  How many images were deleted by digest — the images the fill writes with no tag, where nearly all of a store's weight is.

- **status.replicas.garbageCollection.removedVersions**
  array of strings

  The Deckhouse Platform releases removed.

- **status.replicas.message**
  string

  A human-readable explanation of the reason.

- **status.replicas.node**
  string

  Required value

  The node the replica runs on.

- **status.replicas.phase**
  string

  Required value

  Where the replica is.

  - `Pending` — the replica has not picked the operation up yet.
  - `Waiting` — the replica is held back; `reason` says by what.
  - `Running` — the replica is running the operation.
  - `Succeeded` — the replica finished the operation.
  - `Failed` — the replica did not finish the operation.
  - `Skipped` — the replica left the storage before it started.
  - `Canceled` — the operation was canceled before the replica finished.

  Allowed values: `Pending`, `Waiting`, `Running`, `Succeeded`, `Failed`, `Skipped`, `Canceled`

- **status.replicas.reason**
  string

  A machine-readable reason for the phase.

  - `Waiting`: `Filling` (the cache is being filled), `Pushing` (an image set is being pushed with `d8 mirror push`), `Turn` (another replica is running the operation; or the replica waited a whole window for its turn and waits for the store to go quiet before waiting again).
  - `Failed`: `OperationFailed` (for a `Repair`, also when a digest could not be repaired), `DeadlineExceeded` (by `deadline`, the replica had not picked the operation up, or its store was still being written to), `TimedOut` (the replica did not report a result within 24 hours), `ReplicaLost` (the replica is gone: it stopped holding the collection lease while it collected, or its node was not `Ready` for ten minutes while it waited).
  - `Skipped`: `ReplicaGone`, `NoCache`.
  - `Canceled`: `Canceled`.

- **status.replicas.repair**
  object

  `Repair` results on the replica.

  - **status.replicas.repair.digests**
    array of objects

    One entry per digest of the operation.

    - **status.replicas.repair.digests.digest**
      string

      Required value

      The digest the operation names.

- **status.replicas.repair.digests.message**
  string

  Why it failed, or where the whole copies came from.

- **status.replicas.repair.digests.outcome**
  string

  Required value

  Results:

  - `Intact` — every blob matches its digest; nothing was changed.
  - `Repaired` — broken or missing blobs were replaced with whole copies.
  - `NotHeld` — the replica holds neither an image nor a blob by this digest.
  - `Failed` — a blob could not be replaced; `message` says why. The blob stays as it was.

  Allowed values: `Intact`, `Repaired`, `NotHeld`, `Failed`

- **status.replicas.repair.digests.repairedBlobs**
  integer

  How many blobs were replaced.

- **status.replicas.startedAt**
  string

  When the replica started.

- **status.replicas.verify**
  object

  `Verify` results on the replica.

  - **status.replicas.verify.checkedBlobs**
    integer

    The number of blobs read and compared with their digests.

  - **status.replicas.verify.checkedBytes**
    integer

    Volume of data read in bytes.

  - **status.replicas.verify.corruptedBlobs**
    array of strings

    The blobs whose content does not match their digest, sorted, at most 50.

  - **status.replicas.verify.corruptedCount**
    integer

    The number of broken blobs found, including those beyond the list limit.

  - **status.replicas.verify.missingDigests**
    array of strings

    The digests asked about that the replica holds neither as an image nor as a blob.

- **status.startedAt**
  string

  When the first replica started.

RegistryStorage

Short names: rstor

Scope: Cluster
Version: v1alpha1

Defines how the in-cluster registry storage operates, and reports the state it is actually in.

The object is written by the registry controller and read by the syncer sidecar, which reconfigures `distribution` on the fly. This is the path that makes a credential or certificate change take effect while the Deckhouse operator is down: it never goes through a helm re-render.

It carries a real `spec` rather than being a status-only mirror, because `distribution` has to be told what to do, not merely observed.

Only one object named `registry` is used.

- **spec**
  object

  Required value

  - **spec.airGapRequested**
    boolean

    Whether the configuration asked to run without an upstream, whether or not the upstream has been dropped yet.

    This is the transition window: air-gap asked for, the upstream still held so the cluster keeps working, images arriving through `d8 mirror push` that the syncer never sees. It is what decides that completeness must be measured by reading the store rather than by counting what a copy wrote — a distinction that matters because a pass-through cache writes into its own store on every miss, so the same reading on a caching cluster would count images it merely fetched.

  - **spec.auth**
    string

    The bearer token guarding the storage.

    Sensitive. Authentication is never disabled, not even in an air-gapped cluster where the cache only serves reads.

  - **spec.garbageCollection**
    object

    When and whether a replica reclaims the disk taken by releases the cluster has moved past.

    Needed because nothing else ever removes anything: every release adds a slice of the repository, so a cluster that lives for years fills its store and then stops being able to pull.

    A collection does not interrupt serving: the replica keeps serving every image it holds. It does not start while the store is being filled or a `d8 mirror push` is in progress, and while it runs the replica stops filling its own store. A `d8 mirror push` started during a collection can fail and has to be retried. Only one replica collects at a time, so the others are unaffected.

    - **spec.garbageCollection.enabled**
      boolean

      Whether the controller creates garbage collections on its own: on the schedule, after a new image set, and when the store refuses writes.

      A RegistryOperation created by hand runs either way.

    - **spec.garbageCollection.schedule**
      string

      A five-field cron expression, in UTC.

      Defaults to a night hour, or to the start of the `master` node group’s maintenance window when one is set — that being the time an operator has already declared safe for disruption.

  - **spec.needSync**
    boolean

    How the controller hands the syncers a fill or replication task.

    Set while any replica is short of the expected set: the leader filling from the upstream, or a follower that has not caught up with the leader. Cleared once every replica holds the set; nothing is copied until the set changes again.

  - **spec.publish**
    boolean

    Whether to expose an authenticated write endpoint (Ingress) for `d8 mirror push`.

    Write credentials are separate from read credentials. The endpoint is a second listener of the same registry, served without the cache in front of the store, so publishing it costs the address the cluster pulls through nothing — one listener cannot both proxy an upstream and

accept a push.

- **spec.source**
  object

  The image set that must end up in the storage.

  - **spec.source.bundleRef**
    string

    The name of the image set.

  - **spec.source.expectedDigests**
    integer

    The number of distinct digests the set contains.

    Allowed values: `0 <= X`

- **spec.store**
  object

  Where `distribution` keeps its blobs.

  - **spec.store.path**
    string

    Path on the node. Set by the module, not by the user: `/var/lib/deckhouse/registry` when the master NodeGroup has `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` otherwise.

    Example:

    ```
    path: "/opt/deckhouse/registry"
    ```

  - **spec.store.size**
    string

    The budget of each replica's store, taken from `storage.size` of the ModuleConfig: how much its blobs and uploads in progress may take. Unset means no budget; the reserve kept free on the store's filesystem (see `status.replicas[].store.reserveBytes`) applies either way.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.upstream**
  object

  The primary upstream, projected here by the controller.

  Set means a pass-through cache. Absent means the cache is authoritative (air-gap).

  The controller removes this field **only after the leader is full**, so nodes are never left without a source of images — even if the upstream has already been removed from the configuration by the user.

  - **spec.upstream.auth**
    object

    Where the credentials for this endpoint are kept.

    Named rather than carried. This resource is cluster-scoped and the node agent's role is bound to the `system:nodes` group, so a credential written here would be readable through the API by every kubelet in the cluster — including credentials for registries that node never pulls from. The component that reads this resolves the reference once and keeps the result, so it still works when the API server does not.

    - **spec.upstream.auth.secretRef**
      object

      The Secret holding the credentials, and the key inside it.

      - **spec.upstream.auth.secretRef.key**
        string

        Required value

Key inside it. One Secret holds every credential the module resolved, keyed by what it belongs to.

- **spec.upstream.auth.secretRef.name**
  string

  Required value

  Name of the Secret in the `d8-system` namespace.

- **spec.upstream.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the registry.

- **spec.upstream.host**
  string

  Required value

  The registry host, optionally with a port.

- **spec.upstream.mirrors**
  array of objects

  Additional addresses serving the same content. The syncer pulls from whichever is reachable and stores one de-duplicated set.

  - **spec.upstream.mirrors.auth**
    object

    Where the credentials for this mirror are kept.

    Named rather than carried, for the same reason as the endpoint above: this resource is readable by every node in the cluster. A mirror can need credentials of its own, so it can name a key of its own.

    - **spec.upstream.mirrors.auth.secretRef**
      object

      The Secret holding the credentials, and the key inside it.

      - **spec.upstream.mirrors.auth.secretRef.key**
        string

        Required value

        Key inside it.

      - **spec.upstream.mirrors.auth.secretRef.name**
        string

        Required value

        Name of the Secret in the `d8-system` namespace.

  - **spec.upstream.mirrors.ca**
    string

    A PEM-encoded certificate authority bundle used to verify the mirror.

  - **spec.upstream.mirrors.host**
    string

    Required value

    The mirror host, optionally with a port.

  - **spec.upstream.mirrors.path**
    string

    The repository prefix inside the mirror.

  - **spec.upstream.mirrors.scheme**
    string

    The protocol used to reach the mirror.

    Default: `HTTPS`

Allowed values: `HTTP`, `HTTPS`

- **spec.upstream.path**
  string

  The repository prefix inside the registry.

- **spec.upstream.scheme**
  string

  The protocol used to reach the registry.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **status**
  object
  - **status.allReplicasFull**
    boolean

    Whether every replica holds the expected set, so losing the leader costs nothing.

  - **status.authoritative**
    boolean

    Whether the upstream has been dropped and the cache is now the only source of images.

  - **status.conditions**
    array of objects

    The detailed state of the storage.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      When the condition last changed.

    - **status.conditions.message**
      string

      A human-readable explanation.

    - **status.conditions.observedGeneration**
      integer

      The `spec` generation the condition was set for.

    - **status.conditions.reason**
      string

      Required value

      A machine-readable reason for the condition.

    - **status.conditions.status**
      string

      Required value

      The condition status.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      The condition type.

  - **status.fill**
    object

    Progress of the current fill task.

    - **status.fill.filled**

integer

The number of digests written so far.

- **status.fill.total**
  integer

  The number of digests expected.

- **status.leader**
  string

  The node currently holding the storage leader lease.

  Not a plain election. Leadership is not a symmetric role — the leader is the replication source every follower copies from, and the one whose completeness gates going air-gap. Once some replica holds the whole expected set, a leader that cannot complete the set by itself steps aside for it: one with no upstream, one whose last pass failed, one whose store refuses writes, or one that has stopped renewing the lease. A leader still completing the set from the upstream keeps the lease even if a follower finishes first. While no replica is full, which is where every cluster starts, the lease does not move; with no leader at all, it goes to the replica holding the most verified images.

  Without that rule an air-gapped cluster can stall. `d8 mirror push` arrives through the publication endpoint and always lands on the leader; an incomplete leader beside a full follower (for example, one whose store was lost) has no upstream to fill itself from, the followers replicate its gaps, and the replica holding the images sits idle.

- **status.observedGeneration**
  integer

  The `spec` generation the controller last acted on.

- **status.phase**
  string

  A coarse summary of the fill state.

  - `Idle` — no fill or verification task is running.
  - `Filling` — the leader is being populated.
  - `Verifying` — the contents are being checked against `spec.source`.
  - `Ready` — the leader holds the expected set.
  - `Failed` — a fill or verification task failed. The upstream is **not** dropped in this state.

  Allowed values: `Idle`, `Filling`, `Verifying`, `Ready`, `Failed`

- **status.replicas**
  array of objects

  The per-replica fill state.

  This is the **only** place storage completeness is reported. RegistryNode deliberately carries none of it, since completeness is a property of the storage and not of a node.

  - **status.replicas.address**
    string

    Where this replica answers, as `host:port`.

    Reported by the replica itself so that a follower can replicate from the leader without resolving a node name into an address.

  - **status.replicas.aheadSet**
    string

    The parts of what is ahead of the set that this replica holds as well — the platform version the cluster updates to next, and each module update it is about to apply — as their keys, sorted and comma-separated.

    A part is reported once a copy ahead left it whole. Separate from `verifiedSet` because it is copied separately — after the set, in the background, best-effort — and by part, so that one module that never arrives does not keep the rest from counting. A follower copies the parts its leader holds and it does not, so an update brought to the leader alone, such as one pushed with `d8 mirror push`, reaches the other replicas. Empty when nothing is pending or nothing pending has been copied.

  - **status.replicas.collectedAt**

string

When this replica last reclaimed its disk.

Absent until it has. Reported per replica because each collects its own store, one at a time: a cluster where one replica has not collected for a month while the others have is a different problem from one where none has.

- **status.replicas.collectionError**
  string

  Why the last attempt to reclaim did not finish, empty when it did.

  Separate from `error`, which is about filling. A store that cannot be reclaimed still serves every image it holds, so the two say different things about how worried to be.

- **status.replicas.corruptedBlobs**
  array of strings

  The blobs of this replica’s store whose content does not match their digest, sorted, at most 100. Usually empty.

  They are found during the background check, which reads the whole store once an hour, and by a RegistryOperation of type `Verify` or `Repair` that could not finish the repair. An image that references such a blob is not counted as held: while an image of the set does, the replica is not `full`, and a leader hands the lease to a replica that holds the set whole.

  The module creates a `RegistryOperation` of type `Repair` for these blobs as soon as no other operation runs, and again four hours after a repair that did not fix them.

- **status.replicas.declaredDigests**
  integer

  How big the set **is** — the number `verifiedDigests` is out of.

  Reported by the replica because only the replica computing the set knows its size, and nothing is decided from it. Before it existed the only denominator the status could use was `spec.source.expectedDigests`, which an operator states when declaring air-gap and nobody states otherwise — so a cluster with an upstream showed `filled: 0, total: 0` throughout its longest phase while gigabytes were being copied. Completeness is still `full`, and the air-gap transition is still decided from `spec.source.expectedDigests`: a declaration by an operator is not replaced by a number the cluster computes about itself.

- **status.replicas.error**
  string

  Why the last fill or verification task failed, as reported by this replica’s syncer. Empty when the last task succeeded.

  While the last task did not fail, it also carries what `healthError` reports.

  Safety does not depend on this field: a failed replica is simply not `full`, and the upstream is only ever dropped once the leader **is** full. It exists so that an operator can tell “still filling” from “broken”, which the fill counters alone cannot express.

- **status.replicas.full**
  boolean

  Whether this replica holds the expected set.

  Derived from what the syncer actually **wrote**, never from probing the serve path: on a miss a pass-through cache would fetch the image from the upstream and report success on an empty store. In an air-gapped cluster self-filling is impossible, so reading the catalogue and comparing against `spec.source.expectedDigests` is honest there.

- **status.replicas.healthError**
  string

  Why this replica cannot serve what it holds: its registry has not answered for five minutes, or its store has refused writes for two. Empty while both work.

  The syncer checks both every half a minute on its own, whatever the fill is doing, and clears the error once the registry answers and the store accepts writes again; `full` is left as it was and counts again from then on. A leader reporting it hands the lease over to a replica that holds the whole expected set, if there is one; otherwise it keeps the

lease and only reports the error.

While no replica holds the whole set and none leads, the replicas' syncers choose among themselves which one leads and begins filling: the one that holds the most. They skip a replica reporting this error. They also skip one whose store is at its budget or at the node's reserve (`store.writable` is `false`), unless every replica's store is. An `error` alone, such as a failed copy from a leader that is gone, does not keep a replica from being chosen.

- **status.replicas.images**
  object

  The number of images held by this replica, by state. The images themselves are listed by `d8 k get registryimages`.

  - **status.replicas.images.corrupted**
    integer

    Images, of the set or not, that reference a broken blob.

  - **status.replicas.images.incomplete**
    integer

    Images of the set the replica holds without some of their blobs.

  - **status.replicas.images.missing**
    integer

    Images of the set the replica does not hold at all.

  - **status.replicas.images.outOfSet**
    integer

    Images the replica holds that no release or module of the cluster declares — pulled through the cache on request, or left by an older release.

  - **status.replicas.images.present**
    integer

    Images of the set the replica holds whole.

- **status.replicas.node**
  string

  Required value

  The node the replica runs on.

- **status.replicas.role**
  string

  The replica's role in replication.

  - `Leader` — filled from the upstream, or by `d8 mirror push` in an air-gapped cluster, and hands out replication tasks.
  - `Follower` — filled from the leader, ahead of time, so a leader loss does not mean filling from scratch.

  Allowed values: `Leader`, `Follower`

- **status.replicas.scrubbedAt**
  string

  When the background check last finished reading the whole store and comparing every blob with its digest.

- **status.replicas.source**
  string

  The replica this one is filled from. Empty for the leader.

- **status.replicas.store**
  object

  How much of its node's disk this replica's store takes, and whether it still accepts writes. Absent until the replica has measured it.

  The registry enforces both limits itself, on every write; this is the syncer's account of the

same figures, for the status and the alerts. Sizes are rounded to 16 MiB — used up, free and capacity down — so that a filesystem shared with the rest of the node does not rewrite the status on every pass.

- **status.replicas.store.budgetBytes**
  integer

  `storage.size` of the module configuration. Absent when none is set, and then only the reserve applies.

- **status.replicas.store.dedicatedFilesystem**
  boolean

  True when the store has a filesystem of its own: a disk mounted at the store's directory or at the store path above it, holding neither the kubelet's nor containerd's directory. The replica finds this from its mounts when it starts. Absent when the store shares a filesystem with the node, or when that could not be told: such a filesystem is treated as shared.

- **status.replicas.store.filesystemCapacityBytes**
  integer

  Required value

  Size of the filesystem the store is on, as measured now: a filesystem grown under a running store shows its new size, and the reserve follows it.

- **status.replicas.store.filesystemFreeBytes**
  integer

  Required value

  Free space on the filesystem the store is on. On a node without a dedicated disk it is the filesystem the kubelet and containerd use too.

- **status.replicas.store.reason**
  string

  What writes are refused on, when they are.

  - `BudgetExhausted` — the store holds as much as `storage.size` allows;
  - `ReserveExhausted` — the filesystem under the store is down to the space kept free on it, see `reserveBytes`;
  - `NotWritable` — the registry cannot create files in its tree in the store: `docker/registry/v2` and the directories under it, or the store's directory while that tree does not exist yet. Their owner or mode, the immutable attribute, a read-only mount or a security module stands in the way. No limit is involved, and neither a garbage collection nor a larger `storage.size` helps. The replica's `error` names the directory and the command that fixes it. Takes precedence over either limit.

  Allowed values: `BudgetExhausted`, `ReserveExhausted`, `NotWritable`

- **status.replicas.store.reserveBytes**
  integer

  Required value

  How much of the filesystem the store leaves free, taken of the filesystem's current size rounded up to a whole GiB. On a filesystem shared with the node it is the kubelet's soft eviction threshold (10%, at most 40 GiB) plus a margin for other writers (10%, at most 20 GiB). On a filesystem of the store's own (see `dedicatedFilesystem`) only the margin is kept.

- **status.replicas.store.usedBytes**
  integer

  Required value

  What the store's blobs and uploads in progress take.

- **status.replicas.store.writable**
  boolean

  Required value

  False while the store refuses new writes. Reads are never refused: a full store

serves everything it holds, and a pull it cannot keep is streamed from the upstream. The exception is `reason: NotWritable`, where the registry fails such a pull: its directories refuse it before any limit is checked.

- **status.replicas.syncerStartedAt**
  string

  When the syncer process that wrote this entry started, to the second. Set only once that process has itself counted, copied into or rechecked its store; absent before.

  An entry outlives the process that wrote it. A leader whose pod is recreated on the same node takes the lease back at once, and until the new process reads its store, its entry still says what the previous process found, `full` included, even if the store was wiped with the pod. So the upstream is dropped only on a lease holder's entry stamped no earlier than the start of the `syncer` container running in its Ready pod. Both times come from the clock of that node.

- **status.replicas.totalDigests**
  integer

  How many distinct digests this replica holds **altogether**.

  A different population from `verifiedDigests`, and nothing is decided from it. It is here so that a store holding more than this cluster needs — a bundle pushed with other modules, other editions, attestations — looks like what it is rather than like a shortfall against `spec.source.expectedDigests`.

- **status.replicas.verifiedDigests**
  integer

  How many digests **of the set** were confirmed present.

  The set is what the cluster's releases and the modules it keeps declare — the digests deckhouse-controller knows about — so this number is directly comparable with `full` and is what the air-gap transition is decided from. Anything else the store happens to hold is counted in `totalDigests`.

- **status.replicas.verifiedSet**
  string

  A fingerprint of the set this replica was last found `full` against: the platform's versions and the modules the cluster keeps, each with its version and source.

  `full` is only as good as the set it was measured against, and the set moves — a module is installed, the platform is updated. Every replica reports it with its `full`. The leader compares its own with the set the cluster needs now; when they differ it takes `full` back and verifies before an air-gap transition may rely on it. A `full` for another set counts for nothing when the replicas decide who may lead. Empty when `full` is not set.

- **status.safeToDropUpstream**
  boolean

  Whether the **leader** holds the expected set and nothing on the platform still names the upstream, which together are the gate for going air-gap.

  Deliberately the leader alone and not every replica: followers are filled ahead of time and in parallel, so waiting for all of them would delay the transition without adding safety.

  A full cache does not make up for a remaining reference: an image reference that names the upstream by its own name is not answered by the cache once the upstream is gone. For details, see `upstreamReferences`.

  The leader's `full` counts only when the syncer running on its node now wrote it, after reading its own store (see `replicas[].syncerStartedAt`). After the leader's syncer restarts, this stays `false` until its first pass, normally within two minutes.

- **status.upstreamReferences**
  integer

  Number of objects in platform namespaces whose images still reference the upstream the storage is holding.

  Each object is counted separately: Deployment, DaemonSet, StatefulSet and CronJob templates, pods not started yet, pods that would pull again on their next start, and replicas of the storage itself, which cannot pull their own image from the storage once the upstream is gone. A Deployment in the middle of a rollout adds one for its template and one for each

such pod.

Only the `kube-system` and `d8-*` namespaces are counted. Workloads in other namespaces whose images reference the upstream address do not hold the transition, and once the upstream is removed they have nowhere to pull from. Move them onto the in-cluster address before removing the upstream.

Counted while the leader is full and the upstream is still held. If the field is absent, the count was not taken, and `safeToDropUpstream` is `false`, as with a nonzero count.

RegistryUpstream

Short names: rup

Scope: Cluster
Version: v1alpha1

Declares an additional registry the cluster routes to, on top of the primary one.

This is the supported way for a module, an administrator or a user to bring their own registry. A module creates the object with an `ownerReference` on itself, so the object is garbage-collected together with the module; a GitOps tool owning ModuleConfig never competes for field ownership, because every upstream is a separate object.

Traffic through these upstreams is **always transit**: the agent proxies the pull, substituting credentials and the certificate authority, and streams it through without writing to disk or to the cache. Cache space is finite and reserved for Deckhouse components, all the more so in an air-gapped cluster where the cache is the only source of images.

A `match` already claimed by another object, or one that would shadow the primary path, is rejected rather than silently merged. See `status.conflict`.

Objects are accepted in creation order, starting with the earliest created object. Settings from no more than 256 objects are applied at a time. The combined size of their `spec` fields in JSON format must not exceed 256 KiB. If adding the next object exceeds either limit, its settings are not applied. Creating a new object does not displace a previously accepted object from the node configuration.

With the module disabled or in `Unmanaged` mode, there is no agent on the nodes. The objects are then arbitrated by the same rules by `node-manager`, which writes `status` and hands the accepted ones to the nodes. The nodes apply them directly to the container runtime configuration: the runtime pulls from the upstream and its mirrors without a proxy. A `match` equal to the registry of a `ModuleSource` is accepted: on Immutable nodes the object takes over that registry together with its certificate authority, while nodes configured by bashible, apart from those with containerd v1 described below, keep the registry as the `ModuleSource` configures it. While the module configures no node — it is disabled or in `Unmanaged` mode — nodes with containerd v1 (`Containerd`) do not apply an upstream with `path`, or one whose registry is already configured on the node or named twice in the object. Such an object stays accepted, and the message of its `Accepted` condition names the node groups that do not apply it and why.

- **spec**
  object

  Required value

  - **spec.match**
    string

    Required value

    The registry host intercepted on the nodes.

    Pattern: `^[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?(\.[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?)*(:[0-9]{1,5})?$`

    Length: `1..255`

    Example:

    ```
    match: images.virtualization.example.com
    ```

  - **spec.upstream**
    object

    Required value

Where the intercepted pulls are actually sent.

- **spec.upstream.auth**
  object

  Credentials for the registry.

  Sensitive. Credentials are stored inline in the object rather than referenced from a Secret, so that the node agent stays self-sufficient while the API server is unreachable.

  - **spec.upstream.auth.auth**
    string

    Pre-encoded `base64("username:password")`.

    Takes precedence over `username` and `password` when set. When both forms are set, they must name the same credentials.

    If the caller is not allowed to read the `registryupstreams/sensitive` subresource, the API returns `<omitted>` instead of the value.

    Maximum length: `4096`

  - **spec.upstream.auth.password**
    string

    The password for basic authentication.

    If the caller is not allowed to read the `registryupstreams/sensitive` subresource, the API returns `<omitted>` instead of the value.

    Maximum length: `4096`

  - **spec.upstream.auth.username**
    string

    The username for basic authentication.

    Maximum length: `1024`

- **spec.upstream.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the registry when `scheme` is `HTTPS`.

  Only `CERTIFICATE` blocks are accepted, and each of them must parse.

  Maximum length: `65536`

- **spec.upstream.host**
  string

  Required value

  Registry URL, optionally with a port.

  An IPv6 address is written in brackets, followed by a port.

  Length: `1..255`

  Example:

  ```
  host: registry-vendor.example.com
  ```

- **spec.upstream.mirrors**
  array of objects

  Additional addresses serving **the same** content as the registry, used for failover.

  - **spec.upstream.mirrors.auth**
    object

    Credentials for the mirror.

    Sensitive. See `upstream.auth`.

    - **spec.upstream.mirrors.auth.auth**

string

Pre-encoded `base64("username:password")`.

If the caller is not allowed to read the `registryupstreams/sensitive` subresource, the API returns `<omitted>` instead of the value.

Maximum length: `4096`

- **spec.upstream.mirrors.auth.password**
  string

  The password for basic authentication.

  If the caller is not allowed to read the `registryupstreams/sensitive` subresource, the API returns `<omitted>` instead of the value.

  Maximum length: `4096`

- **spec.upstream.mirrors.auth.username**
  string

  The username for basic authentication.

  Maximum length: `1024`

- **spec.upstream.mirrors.ca**
  string

  A PEM-encoded certificate authority bundle used to verify the mirror.

  Maximum length: `65536`

- **spec.upstream.mirrors.host**
  string

  Required value

  Mirror URL, optionally with a port.

  An IPv6 address is written in brackets, followed by a port.

  Length: `1..255`

- **spec.upstream.mirrors.path**
  string

  The repository prefix inside the mirror.

  Maximum length: `1024`

- **spec.upstream.mirrors.scheme**
  string

  The protocol used to reach the mirror.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.upstream.path**
  string

  The repository prefix inside the registry.

  Maximum length: `1024`

  Example:

  ```
  path: "/virt"
  ```

- **spec.upstream.scheme**
  string

  The protocol used to reach the registry.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **status**
  object
  - **status.accepted**
    boolean

    Whether the controller compiled this upstream into the per-node layout.
  - **status.conditions**
    array of objects

    The arbitration result with a machine-readable reason.

    - `Accepted` — whether the upstream is part of the per-node layout.
    - **status.conditions.lastTransitionTime**
      string

      Required value

      When the condition last changed.
    - **status.conditions.message**
      string

      A human-readable explanation.
    - **status.conditions.observedGeneration**
      integer

      The `spec` generation the condition was set for.
    - **status.conditions.reason**
      string

      Required value

      A machine-readable reason for the condition.
    - **status.conditions.status**
      string

      Required value

      The condition status.

      Allowed values: `True`, `False`, `Unknown`
    - **status.conditions.type**
      string

      Required value

      The condition type.
  - **status.conflict**
    string

    Reason for rejecting the object: the `match` value is already used by another object, `match` intercepts requests to the primary image registry (Primary), a field in `spec` is invalid, or adding the object exceeds the limit on the number of accepted objects or the combined size of their `spec` fields.

    Empty when the object is accepted.
  - **status.observedGeneration**
    integer

    The `spec` generation the controller last acted on.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

⚠ The module lifecycle stage: Preview

How does the migration to the registry module work?

There are two ways a Deckhouse Platform (DP) cluster's image pull path can be managed:

- **the v1 implementation** (DP up to and including 1.77) — the `registry` section of the `deckhouse` ModuleConfig, with the modes `Unmanaged`, `Direct`, `Proxy`, and `Local`;
- **the current implementation** — this module, configured through the `registry` ModuleConfig.

The migration is the handover of the pull path from the v1 implementation to the module. It is not started by hand and has no command of its own: the module takes over automatically after the cluster is upgraded to DP 1.78. The only condition is that by then the v1 implementation must have released the pull path. Both implementations configure the same thing on every node — which registry the container runtime pulls from and with which credentials — so they never manage a cluster at the same time.

What has to be done for the handover, and when, depends on the mode the v1 implementation is running in. The mode is set in the `settings.registry.mode` parameter of the `deckhouse` ModuleConfig:

```
d8 k get mc deckhouse -o jsonpath='{.spec.settings.registry.mode}'
```

Empty output means `Unmanaged` mode: the registry settings of this cluster have never been touched. Such a cluster — like any other cluster in `Unmanaged` — needs no action for the migration: the handover happens on its own.

Mode	What to do	When
`Unmanaged`	Nothing — the handover happens on its own	—
`Direct`	Configure the module: `mode: Managed` with `primary.upstream`	Before the upgrade
`Proxy`	Switch the cluster to `Unmanaged`	Before the upgrade
`Local`	Follow the procedure for air-gapped clusters	Before the upgrade

✖ Prepare the cluster **before** the upgrade: upgrading to DP 1.78 is blocked while the cluster is running in `Proxy` or `Local` mode, or in `Direct` mode without a configured `registry` ModuleConfig. DP 1.78 contains neither the components of the v1 implementation nor the code that switches its modes, so all preparation happens on DP 1.77.

`Direct` mode needs no mode switch. The module's settings are deliberately accepted already in DP 1.77: written before the upgrade, they are stored without effect and take effect on the module's first reconciliation after it.

On nodes with containerd v1, a migration from or through `Unmanaged` restarts containerd. See Does the migration restart containerd?.

The cluster records which implementation is actually running (see the next question). While the handover is impossible, the upgrade to DP 1.78 is blocked, and the error message says what to do.

Which implementation is my cluster running?

When the module takes over the pull path, it records that in the `registry-v2-switch` secret. Check whether the secret exists:

```
d8 k -n d8-system get secret registry-v2-switch >/dev/null 2>&1 \
  && echo "current implementation" || echo "v1 implementation"
```

If the cluster is still on the v1 implementation, the module reports the reason on every reconciliation

and fires the `D8RegistryMigrationPending` alert. To see what it is waiting for:

```
d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
```

On DP 1.78 the update deletes the `registry-state` secret. The mode is then still shown by the node configuration the v1 implementation left in the `registry-bashible-config` secret. This command prints only the mode and the proxy endpoints, without credentials:

```
d8 k -n d8-system get secret registry-bashible-config -o jsonpath='{.data.config}' | base64 -d \
  | awk '/^(agent|mode):/ {print; next} /^proxyEndpoints:/ {p=1; print; next} p && /^
*- / {print; next} {p=0}'
```

An `agent:` line in the output, or the `registry.deckhouse.io/installer-config-sha256` annotation on the secret, means the configuration was written by the current implementation or by the installer of DP 1.78 or later, not by the v1 implementation. The annotation counts only while the configuration is unchanged since the installer wrote it.

How do I migrate from Unmanaged mode?

In `Unmanaged` mode the v1 implementation does not manage the pull path, so there is nothing to prepare: the handover happens on its own.

1. If the cluster is switching to `Unmanaged` from another mode, wait for the transition to complete. The status must show `mode: Unmanaged` with no pending target mode:

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

2. Upgrade the cluster to DP 1.78 (or, if it is already on it, just wait for the next reconciliation). The module takes over automatically, and the pull source does not change: the module's default mode is also `Unmanaged`, so the cluster keeps pulling from the same registry as before. On containerd v1 nodes, however, the takeover restarts containerd node by node if the v1 implementation left a node registry configuration, and enabling `mode: Managed` in the next step restarts it (a second time, if the takeover already did; see Does the migration restart containerd?).

3. To have the module manage the pull path, set `mode: Managed` in the `registry` ModuleConfig and specify the registry to pull from. A ready-made configuration for your cluster is published in the `registry-suggested-config` secret — see enabling the module.

How do I migrate from Direct mode?

In `Direct` mode the nodes pull images through an in-cluster address served by the v1 implementation's proxy. The module serves the same address, so the migration is a direct handover of that address: no switch through `Unmanaged` is needed, and no components restart.

1. Configure the module before the upgrade — without this configuration the upgrade is blocked. Take the values from the `registry.direct` section of the `deckhouse` ModuleConfig: `imagesRepo` splits into `host` and `path`, the credentials are the same `license` (or `username`/`password`), plus `ca` if the registry requires one:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: registry
   spec:
     enabled: true
     version: 1
     settings:
       mode: Managed
       primary:
         upstream:
           scheme: HTTPS
           host: registry.deckhouse.io
           path: /deckhouse/ee
           auth:
             license: <LICENSE_KEY>
   ```

DP 1.77 stores these settings without acting on them, so nothing changes in the cluster until the upgrade.

2. Upgrade the cluster. The handover happens on the module's next reconciliation, and pulls keep working throughout: the Service and the proxy of the v1 implementation keep serving the in-cluster address until the module's agent has taken it over on every node, and only then does the controller remove them.
3. Track the progress:

   ```
   d8 k -n d8-system get secret registry-v2-switch >/dev/null 2>&1 && echo "handed over"
   d8 k get registrynode -o custom-columns='NODE:.metadata.name,READY:.status.reconciled,SERVING:.status.proxyListening'
   ```

   For reference, the order and rough scale to expect: the handover is recorded a couple of minutes after the new version starts, the node agent appears on the nodes several minutes later, and the objects of the v1 implementation are removed shortly after that. Pulls keep working at every point in between.

How do I migrate from Proxy mode?

`Proxy` mode keeps a proxy of its own, with its own certificates, on every node — state the module cannot adopt. So the cluster must first be switched to `Unmanaged`, and this can only be done before the upgrade.

1. In the `deckhouse` ModuleConfig, set `registry.mode: Unmanaged`, keeping the same registry address and credentials — see the mode switching examples. All nodes are reconfigured to pull directly from the external registry, so the caching `Proxy` provided is lost until you bring it back in step 4.
2. Wait for the transition to complete — `mode: Unmanaged` with no pending target mode. A cluster caught mid-transition cannot be migrated, and the status shows which mode it is still heading to:

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

3. Upgrade the cluster. The handover happens on the module's next reconciliation, and the pull source does not change: in `Unmanaged` mode the module does not manage the pull path either. On containerd v1 nodes, however, the takeover restarts containerd node by node, and enabling `mode: Managed` in the next step restarts it again (see Does the migration restart containerd?).
4. To get in-cluster caching back, set `mode: Managed` with `storage.cache: true` and the same upstream registry. Note that the module's cache is built differently from `Proxy`: a single store with replicas on the master nodes instead of a proxy on every node. Before enabling it on a cluster with little free disk space on the control-plane nodes, read how the cache is filled and cleaned up.

How do I migrate an air-gapped cluster from Local mode?

In `Local` mode the cluster's registry lives inside the cluster itself, and there is no external one. The standard procedure does not apply here: it goes through `Unmanaged`, where every node pulls directly from an external registry — and this cluster has none.

The way through is to give the cluster an external registry temporarily: run it in the same cluster, but outside the DP namespaces; switch the cluster to it; upgrade; let the module's storage fill up; then remove it. The procedure has been verified end to end on a test cluster.

Disk requirements

Before starting, make sure the control-plane nodes have free space for **four image sets**. Three copies exist at the peak — the `Local` store, the temporary registry, and the module's storage being filled — and the fourth set is headroom the node must not run out of.

To get a sense of the scale, take an image set on the order of 13 GB (platform with modules): it occupies about 21 GB in the temporary registry (two releases — the one the cluster runs and the one it moves to) and grows the store from 13 to 21 GB, on top of the node's own image cache and the system. A 100 GB control-plane node peaks at around 38 GB that way, while a 50 GB node runs into pod eviction.

The headroom is critical. When free disk space on a control-plane node runs low, kubelet starts evicting pods and deleting images it considers unused — and in a cluster whose registry runs inside it, the

deleted image can be the registry's own. The store is then left with no process to serve it, and the node with nowhere to pull that image from. In the test this deadlock lasted 99 minutes and was resolved only by loading images onto the node manually. Plan the disk space in advance and monitor it while the migration runs.

The images already on the control-plane disks are adopted by the module's storage — verified in place rather than downloaded again (see step 6). What adoption cannot do is merge two different releases into one set: the store ends up holding both the release the old cluster ran and the one the new cluster runs, which is where the peak size comes from.

Procedure

Steps 1–4 are performed before the upgrade, on DP 1.77, which still contains the v1 implementation.

1. Run a temporary OCI registry in a namespace of your own. Any registry implementation will do, but it must serve TLS with a certificate the cluster can verify. DP must not manage it: whatever happens to the module's own objects, the temporary registry must keep working.

   The registry must be reachable **at one address from two places**: in step 3 the nodes pull from it, and in step 6 the module's syncer reads it from inside a pod. The certificate must cover the chosen address. Options tested on one cluster:

Address	From a node	From a pod
`hostNetwork` port at the node's own IP (`<NODE_IP>:5000`)	works	works
Service name (`<SERVICE>.<NAMESPACE>.svc:<PORT>`)	does not resolve	works
NodePort at a node's IP	works	fails: `operation not permitted`

   Use the first option: run the registry with `hostNetwork: true` on one node, address it by that node's IP, and include the IP in the certificate's SAN. The whole migration then stays on a single address. A Service name looks tidier but breaks in step 3: the container runtime on a node does not resolve cluster DNS.

2. Load the image set into the temporary registry: run `d8 mirror pull` on a machine with access to the DP registry, then `d8 mirror push` into the temporary one. This is the copy the disk budget above accounts for.

3. Point the v1 implementation at the temporary registry and switch it to `Unmanaged`: in the `deckhouse` ModuleConfig, set `registry.mode: Unmanaged` together with the address, the CA certificate, and the credentials of the temporary registry. The nodes start pulling from it. The `Local` store leaves the pull path, but its data stays where it is — under `/opt/deckhouse/registry` on the control-plane nodes.

4. Make sure the transition is complete and images really are pulled from the temporary registry — the rest of the migration rests on this step. The checks are the same as for any `Unmanaged` cluster:

   ```
   d8 k -n d8-system get secret registry-state -o jsonpath='{.data.state}' | base64 -d | head
   ```

5. Upgrade the cluster to DP 1.78. The handover happens on the module's next reconciliation; the cluster keeps pulling from the temporary registry throughout. On containerd v1 nodes the takeover restarts containerd node by node, and enabling `mode: Managed` in the next step restarts it again (see Does the migration restart containerd?).

6. Enable the module: `mode: Managed` with `storage.cache: true`, the temporary registry as `primary.upstream` — **and `storage.source` in the same edit**. The module's storage starts on the same host path the `Local` store used, so the images already on the disks are adopted: the fill verifies them and downloads only what is missing.

   Do not postpone specifying `storage.source`: without it, the upstream cannot be removed in the next step. A `Managed` configuration without `primary.upstream` is only accepted when `storage.source` is set, so the next step would be rejected with `'storage.source' is required when 'primary.upstream' is not set`.

   In `storage.source`, `bundleRef` is a name for the image set, and `expectedDigests` is the number of distinct digests in it. Count them in the bundle you pushed:

```
for tar in <BUNDLE_DIR>/*.tar; do tar -xOf "$tar" --wildcards '*index.json'; done |
  jq -r '.manifests[]?.digest' | sort -u | wc -l
```

Setting `storage.source` does not restart the storage: it is not part of the registry process's configuration. The registry restarts in the next step, when the upstream is removed.

7. Wait until the storage reports that it holds the whole set — `phase: Ready` with `safeToDropUpstream: true` — and remove `primary.upstream` from the `registry` ModuleConfig. The cluster is air-gapped again, now on the module. After a storage replica restarts, `safeToDropUpstream` can stay `false` until the leader's syncer has read its store, normally within two minutes, or about ten minutes if that first reading fails:

   ```
   d8 k get registrystorage registry -o jsonpath='{.status.phase}
   {.status.safeToDropUpstream}{"\n"}'
   ```

   Removing the upstream restarts the registry process. If the restarted registry spends more than a few tens of seconds reading its store, the RegistryStorage resource reports `Failed` with an error about reading the image set. This is expected; wait for it to return to `Ready`.

8. Delete the temporary registry and reclaim its disk space.

Does the migration restart containerd?

On nodes with containerd v1 (`cri.type: Containerd`), it does. Nodes with containerd v2 always take their registry settings from the `/etc/containerd/registry.d` directory, and containerd is not restarted on them.

On a containerd v1 node, `/etc/containerd/config.toml` points containerd at `/etc/containerd/registry.d` only while the cluster has a node registry configuration, the `registry-bashible-config` secret. Without that secret, the file describes the registry with its own `mirrors` and `configs` blocks. In `Unmanaged` mode the module removes the secret, including the one the v1 implementation left behind. A change of `config.toml` restarts containerd, and kubelet after it. Therefore, on containerd v1:

- every switch between `Unmanaged` and `Managed`, in either direction, restarts containerd once. From `Managed` to `Unmanaged`, the restart comes when the module stops serving, that is once no workload names the in-cluster address `registry.d8-system.svc:5001` any more, not when the ModuleConfig is changed;
- a cluster where the v1 implementation left the secret restarts it twice on the way through `Unmanaged`: when the module takes over after the upgrade, and again when `mode: Managed` is enabled. That is a cluster switched from `Proxy` or `Local` to `Unmanaged` with a registry address before the upgrade, and any other `Unmanaged` cluster with a registry address;
- a migration from `Direct` does not restart it: the node configuration is kept throughout.

To check whether the secret is there, run `d8 k -n d8-system get secret registry-bashible-config`.

containerd is not restarted on all nodes at once. The change reaches them as a regular node update, which waits for approval, and approval is granted one node at a time in each NodeGroup by default (see `update.maxConcurrent`). Running containers survive the restart. On the node being updated, image pulls in progress fail and kubelet retries them, and open `exec`, `attach`, `port-forward` and `logs -f` sessions are cut.

What do the registry alerts mean?

Apart from the two about the node agent, `D8RegistryStorageLeaderAbsent` in an air-gapped cluster and `D8RegistryMigrationPending`, none of these alerts mean that the cluster has stopped pulling images. Most of them report a state where everything works, but not the way the configuration asks — a state that is easy to miss otherwise.

`D8RegistryMigrationPending`
: The module has not taken over from the v1 implementation, used up to and including DP 1.77, and the alert's reason says why. If the reason names the `Direct` mode, configure the module as described in migrating from Direct mode: the module then takes over on its own. Any other reason (the `Proxy` or `Local` mode, a transition, no recorded mode, or a state that cannot be read) means a state that is not supported. A cluster gets there only through an update that bypassed the `registryImplementation` release requirement: a forced release, an edited image, or a jump from a release older than DP 1.77.1. DP 1.78 does not deploy the v1 implementation, and the update removed the components that managed its node services on the masters, so those services were stopped or keep running with nothing to manage them. Images already on a node

keep working; pulling any other image may fail. Do not change the `registry` or `deckhouse` ModuleConfigs and do not delete the `registry-*` secrets in `d8-system`: those secrets keep the current implementation from being deployed next to what the v1 implementation left. Contact Deckhouse Platform technical support. See also which implementation the cluster is running.

`D8RegistryConfigInvalid`
: The configuration was rejected; the cluster keeps working with the previous one. The reason is in `.status.conditions` of the `registryconfig/registry` resource.

`D8RegistryNodeNotConverged`
: The agent on some nodes has not applied the configuration it was given. Those nodes keep pulling with their old configuration, and further changes will not reach them either.

`D8RegistryNodeRunningFromDisk`
: The agent on some nodes could not get a usable configuration from the API server and routes pulls using the copy stored on disk. Most often the credentials the configuration references cannot be read; a read of the configuration that failed while the status write after it went through, as on a short API server outage, looks the same. This fallback works as designed: pulls on those nodes succeed, so nothing else would report the problem — meanwhile their configuration can lag arbitrarily far behind the cluster's. The agent reports the fallback through the API server, so the alert does not cover an agent that cannot write there: a node cut off from the API server shows as `NotReady` instead, and an agent refused access on a `Ready` node is not reported at all.

`D8RegistryAgentNotReady`
: The agent on a node has not been ready for 15 minutes: it fails on start, does not answer, or is not running on the node at all. Until it is back, containerd on that node pulls without it: platform images and images from registries that need the cluster's credentials or CA fail, other images bypass the cache. See where to look.

`D8RegistryAgentRestartingTooOften`
: The agent on a node has restarted five times or more in an hour. Each restart drops the pulls in progress on the node. The container's last state and the log of the previous run say whether it exited on its own or was restarted for not answering.

`D8RegistryStorageIncomplete`
: Some cache replicas do not hold the whole expected image set. While an upstream is configured this does not affect pulls, but the cluster would not survive losing the upstream — and this is what blocks a transition to air-gap. A replica whose store the registry cannot write to (`NotWritable`) is reported by `D8RegistryStoreWritesRefused` instead.

`D8RegistryAirGapTransitionHeld`
: The upstream was removed from the configuration, but the module keeps using it: either the cache cannot serve the cluster alone yet, or the cache leader's report was not written by the `syncer` running on its node now (`syncerStartedAt` in the RegistryStorage status), or workloads in the `kube-system` and `d8-*` namespaces still reference the upstream (`upstreamReferences`). This is the safe outcome: dropping the upstream at that point would leave the nodes with nowhere to pull from. The alert does not resolve on its own if the cache has stopped filling, a rollout onto the in-cluster address is stuck, the leader's storage pod is not Ready, or that pod runs a `syncer` from before `syncerStartedAt` existed (during a module update).

`D8RegistryUpstreamProbeFailing`
: A change to the primary upstream was rejected, and the cluster keeps using the last working one. The `outcome` label tells the problems apart: `unreachable` — network or the registry itself; `auth` — usually an expired license key; `sentinel` — the registry responded and accepted the credentials, but does not contain the DP images (usually a wrong repository path).

`D8RegistryUpstreamRejected`
: A RegistryUpstream resource was not accepted, so pulls for the registry it names are not intercepted. The `reason` label says whether it conflicts with the primary registry or with another resource claiming the same name.

`D8RegistryStorageNotReclaimed`
: No replica has run garbage collection for a week. Collection is the only thing that removes data from the store, so if it has stopped, the disk will eventually fill up. See what cleans the cache up.

`D8RegistryStoreCorrupted`
: A replica holds blobs whose content does not match their digest, and the repair the module started on its own has not fixed them for two hours. Pulls of the affected images from that replica fail. See how to check and repair the images.

`D8RegistryStaleCacheData`
: A node holds cache data that nothing uses. See how to remove it.

`D8RegistryStoreWritesRefused`
: The cache on a master has stopped accepting writes: its store reached `storage.size` (`BudgetExhausted`) or the space kept free on its filesystem (`ReserveExhausted`). The node is safe and the cache keeps serving what it holds; what stops is filling it. See what happens when the store is full. `NotWritable` means something else: the registry cannot create files in its tree in the store (`docker/registry/v2` and below, or the store's directory while that tree does not exist yet), because of their owner or mode, an attribute, a read-only mount or a security module. Neither garbage collection nor a larger `storage.size` helps. The replica's `error` in the RegistryStorage status names the directory and the command that fixes it.

`D8RegistryStoreNearlyFull`
: The store on a master holds over 85% of `storage.size`. The next release may not fit.

`D8RegistryStoreUnbounded`

The cache is on and `storage.size` is not set, so the only limit is the space kept free on the store's filesystem.

`D8RegistryControllerDown`
No replica of `registry-controller` is ready and answering Prometheus. Pulls are not affected, but no configuration change reaches the nodes, and most other alerts of the module are silent while this one fires: they read what the controller exports. `D8RegistryStorageLeaderAbsent` and the migration and drain alerts do not depend on the controller and keep working.

`D8RegistryStorageLeaderAbsent`
No ready cache replica reports holding the leader lease. Nothing fills the cache, the other replicas have nobody to replicate from, and `d8 mirror push` has nowhere to go. The nodes still route pulls to the replica named as leader: with an upstream configured pulls are slower, in an air-gapped cluster images the nodes do not already hold cannot be pulled. If the Lease `registry-storage` in `d8-system` keeps being renewed, the leader is alive but its report is not scraped: either its pod is not ready, then find out which of its containers is not ready, or the scrape of that pod fails, then look at the log of its `kube-rbac-proxy` container. If the master that held the lease is lost and is not coming back, delete its Node object; otherwise look at the `error` the replicas report in the RegistryStorage status and at their `syncer` logs. A cache replica that is merely not ready is reported by `KubernetesStatefulSetReplicasUnavailable` of the `extended-monitoring` module, if it is enabled.

`D8RegistryStorageUpdateBlocked`
The cache has a new revision, but for an hour the controller has been unable to replace the next replica. Existing replicas remain running, but the cache update cannot proceed. The `reason` label says what the controller is waiting for and whether anything needs doing. See how the cache is updated.

How do I remove leftover cache data from a node?

When the cache is turned off, the data in the cache directory is intentionally kept: if the cache is turned back on, it refills from what is already on disk instead of downloading everything again — over a slow link that saves hours. Deleting the data automatically would make turning the cache off irreversible, so the module leaves the decision to you: the agent measures the leftover data and fires `D8RegistryStaleCacheData`.

Check how much space the data takes:

```
d8 k get registrynodes -o custom-columns=\
NODE:.metadata.name,STALE:.status.staleStorageDataBytes
```

If you are not going to turn the cache back on, remove the contents of the cache directory on the node: `/var/lib/deckhouse/registry` on a node of a NodeGroup with `systemType: DeckhouseEngine`, `/opt/deckhouse/registry` on other nodes. Keep the directory itself: the agent has it mounted, and once it is removed the agent no longer sees the node's directory, so data a cache leaves there later goes unreported until the agent restarts. To do this, follow these steps:

1. Open a root shell on the node over SSH. A node of a NodeGroup with `systemType: DeckhouseEngine` has no SSH: start a debug pod on it and switch to the node's root filesystem:

   ```
   d8 k debug node/<NODE> -it --image=<IMAGE>
   chroot /host
   ```

2. Remove the data. The command checks both directories and keeps the directories themselves:

   ```
   for dir in /opt/deckhouse/registry /var/lib/deckhouse/registry; do
     [ -d "$dir" ] && du -sh "$dir" && find "$dir" -mindepth 1 -delete
   done
   ```

The cache keeps growing. What reclaims it?

Garbage collection, which every storage replica runs on its own store. The module starts it on a schedule, once the cache holds a new image set, and when the store refuses writes; it can also be started by hand.

It is the only mechanism that removes anything from the store. Every DP release adds new images, so without collection the store of a long-lived cluster eventually fills up and stops accepting writes — and an air-gapped cluster with a full store cannot be updated.

Collection removes the images of the releases the cluster has moved past. It keeps:

- the deployed release and the previous one, so that a rollback does not have to re-download

anything;
- everything newer than the deployed release — an update in progress or, in an air-gapped cluster, a release pushed on purpose;
- every tag that is not a version: release channel names such as `stable`, floating tags, anything pushed by hand. The collector cannot know what these mean, so it does not touch them.

It also removes blobs that no image references, such as those left by an interrupted fill or by a collection whose cleanup failed. A collection removes them even when there are no images to delete.

The collector is deliberately cautious. In an air-gapped cluster, deleting a blob that is still needed is unrecoverable without another `d8 mirror push`, while keeping an unneeded one merely costs disk space. So a run that cannot determine what to keep — for example, when no deployed release is found — does nothing at all.

To check the collection status and schedule, use these commands:

List the recent collections — what started them, how they went, and how much they freed. A finished collection is kept for a day:

```
d8 k get registryoperation -o wide
```

Check the last collection time on each replica, and any errors:

```
d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq \
  'map({node, collectedAt, collectionError})'
```

Check the current collection schedule:

```
d8 k get registrystorage registry -o jsonpath='{.spec.garbageCollection}' | jq
```

When it runs, and what happens during a run?

The registry’s collector first computes the set of reachable blobs and then deletes the rest, so a blob written between those two steps and not yet referenced by a manifest would be deleted. A replica therefore collects only when its store is not being written to:

- a collection does not start while the cache is being filled or replicated, or while a `d8 mirror push` is in progress — it waits until the writing is over;
- the replica looks at its store again when its turn comes: if a fill or a push started while it waited, it gives the turn back and waits again;
- it looks once more right before removing blobs: if a push has started by then, the blobs are left for the next collection;
- while it collects, the replica stops filling and replicating its own store.

The replica keeps serving all the images it holds and fetching cache misses from the upstream. Do not run `d8 mirror push` while a collection is running: check `d8 k get registryoperation` first. A push that starts during the removal of blobs can fail with a visible error, and it can also report success and leave an image incomplete: a layer the store already held is not uploaded again, and the collection can remove it. Such an image fails to pull later with a missing blob. If a push overlapped a collection, run it again after the collection finishes.

Only one replica collects at a time; the others work normally.

The schedule is in UTC. By default, collection is scheduled at a night hour — or, if the `master` node group has a maintenance window, at its start, since that hour is already declared safe for disruption. To set your own schedule, use the `storage.garbageCollection` parameter:

```
spec:
  settings:
    storage:
      garbageCollection:
        schedule: "0 2 * * Sun"
```

A cron expression that cannot be parsed is rejected rather than guessed at: collecting at an unexpected hour is worse than not collecting. For the same reason a changed schedule counts from the moment it changed: a firing of the new schedule that has already passed is not made up for. The same applies after collection is turned back on.

Collection also starts once every replica holds a new image set: after a Deckhouse Platform update, or

after a module is enabled or updated. That is the moment the images only the previous set needed become garbage. Disabling a module does not change the set: its images are kept for when it is enabled again.

A store that refuses writes — over its budget, or down to the space kept free on its filesystem — can never take in a new set, so it is collected without waiting for one: once per deployed Deckhouse Platform release, and again after an update or after the store has accepted writes in between. If that collection fails, it is tried again four hours after it ended.

Every collection is a RegistryOperation object of type `GarbageCollection`, whoever started it. Only one runs at a time: a firing of the schedule that comes while another collection is running is skipped, and a collection for a new image set waits for the running one to finish.

Is a collection worth running now?

Each storage replica estimates how much a collection of its store would free and exports the estimate to Prometheus metrics:

- `d8_registry_storage_reclaimable_bytes`: how many bytes a collection would free now. Zero means a collection would free nothing.
- `d8_registry_storage_reclaimable_estimated_timestamp_seconds`: when the estimate was made.

Both carry the `node` label of the replica. Neither is exported before the replica's first estimate.

A replica makes the estimate:

- when the image set the cluster needs changes;
- after a collection on that replica;
- every 30 minutes otherwise, and not more often than once in 5 minutes.

Estimate is not performed:

- during the filling of the cache;
- while uploading images to the storage ('d8 mirror push');
- during garbage collection.

Therefore, when viewing an estimate, pay attention to its time. Storage that refuses to write is evaluated as usual: this is when garbage collection is most needed.

To see the estimate for every replica, run the following query in Prometheus:

```
d8_registry_storage_reclaimable_bytes
```

How to run garbage collection manually?

To run garbage collection manually, create a RegistryOperation object of type `GarbageCollection`:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: gc-manual-
spec:
  type: GarbageCollection
  cause: Free space before pushing a new release.
EOF
```

It runs even when automatic collection is turned off. Replicas collect one at a time, exactly as on schedule. A replica starts only when its store is not being written to: while the cache is being filled or a `d8 mirror push` is in progress, the replica waits, and the reason is shown in `status.replicas[].reason`. A store that refuses writes is not waited for: a fill cannot go on until the collection frees room. A replica still held back an hour after the operation was accepted fails with the reason `DeadlineExceeded`. An operation created while another one is running fails with the reason `AnotherOperationActive`.

Follow the operation and see what it removed:

```
d8 k get registryoperation <NAME> -o jsonpath='{.status}' | jq \
  '{phase, reason, message, garbageCollection}'
```

To cancel an operation, including one a replica is already running, set the cancel label on it:

```
d8 k label registryoperation <NAME> registry.deckhouse.io/cancel=true
```

What was deleted before the cancellation stays deleted.

If a replica's `syncer` container stops answering while the replica collects or waits for its turn, for example because its master went `NotReady`, the replica fails with the reason `ReplicaLost`, so that the operation does not hold every collection after it for a day:

- a replica that was collecting, once it has not held the Lease `registry-storage-gc` in the `d8-system` namespace for five minutes;
- a replica that was waiting, once its node has not been `Ready` for ten minutes. A master that is back sooner keeps its replica's turn.

A replica that waits on a node that stays `Ready` while its `syncer` does not answer is released by the cancel label, by the 24-hour timeout (the reason `TimedOut`), or by deleting the operation:

```
d8 k delete registryoperation <NAME>
```

Turning it off

To disable automatic garbage collection, set the value to `false` for the `storage.garbageCollection` parameter:

```
spec:
  settings:
    storage:
      garbageCollection:
        enabled: false
```

This only makes sense with a disk large enough that unbounded growth of the store never becomes a problem. Note that `D8RegistryStorageNotReclaimed` still fires a week after the last collection: from the outside, "turned off" and "silently stopped" look the same.

Garbage collection created [manually] (#how-to-run-garbage-collection-manually) is performed even if automatic is disabled.

What happens when the cache's store is full?

Each master's store has two limits, and the registry checks both on every write:

- the budget, `storage.size` in the `registry` ModuleConfig — how much the store may hold;
- the reserve — how much of the filesystem under the store must stay free. On a filesystem shared with the node it is the kubelet's soft eviction threshold (10% of the filesystem, at most 40 GiB) plus a margin (10%, at most 20 GiB). On a disk of the store's own (`dedicatedFilesystem: true` in the replica's `store`) it is the margin alone. Both are taken of the filesystem's current size, so a disk grown under a running store moves the reserve without a restart. The reserve applies whatever `storage.size` says, including when it is not set.

The reserve is taken from the full size of the filesystem, while free space is counted the way the kubelet counts it: without the blocks the filesystem keeps for root. On ext4 these are 5% by default, so on a filesystem under 200 GiB the store and everything else on it get about 75% of the disk rather than 80%. On a filesystem dedicated to the store, the root reservation can be lowered with `tune2fs -m`.

When either limit is reached, the store refuses new writes and keeps serving everything it holds:

- a fill or a replication stops. Why the store refuses is in the replica's `store` field (`writable: false` and `reason`, see below), and `.status.replicas[].error` holds the error of the fill or replication that stopped;
- `d8 mirror push` gets `507 Insufficient Storage` with the reason and the way out, before the upload's data is sent. A blob larger than the space left can still be refused in the middle of its upload, with `500`; push again once there is room;
- a cache miss is served from the upstream while there is one, so the nodes do not notice: the layers the store cannot keep are streamed through without being stored, and a manifest the store

cannot write fails at the store, so the agent takes the image from the upstream. In an air-gapped cluster a release that is not in the store cannot be installed.

The store keeps the deployed release and the previous one, for a rollback, together with the modules the cluster runs and the catalog of modules the module sources offer. It also needs room for the next release:

- a replica copies the next release ahead of the update only while the room it has left in the budget, and on the filesystem above the reserve, is at least a fifth of the budget (or of the filesystem, when `storage.size` is not set); otherwise the copy ahead is skipped, and the release is filled once it is deployed. With an upstream, this applies to every replica;
- in an air-gapped cluster, the next release arrives with `d8 mirror push` on the leader, and the garbage collection does not delete the deployed or the previous release to make room for it. The followers copy it from the leader under the same rule, so a follower short of room gets the release only once it is deployed.

So size `storage.size` for three releases together with the modules and the catalog. The store holds the union of their images: a layer the releases share is stored once.

Nothing is deleted to make room. Space is freed by the garbage collection, or made by raising `storage.size` or giving the store more disk.

To see where each replica stands:

```
d8 k get registrystorage registry -o json | jq '.status.replicas[] | {node, store}'
```

`writable: false` with `reason: BudgetExhausted` means `storage.size` is reached: raise it if the disk has room, or let the collection reclaim what no deployed release needs. `reason: ReserveExhausted` means the disk is. On a filesystem shared with the node, free space on the node or move the store to a disk of its own. On a disk of its own, grow the disk or let the collection reclaim space. `reason: NotWritable` is not a limit: the registry cannot create files in the store, and `.status.replicas[].error` names the directory and the command that fixes it.

The store’s disk did not mount. What happens?

If the store was on a disk of its own (`dedicatedFilesystem: true`) last time and that disk is not mounted now, for example after a reboot, the store starts on the node’s own filesystem and fills there. The replica reports this in `.status.replicas[].error` for as long as the disk is missing, and keeps reporting it after the store has filled. A replica with an error does not count as full:

- it does not stay the leader while another replica is full: that replica takes over;
- if it is the leader and no other replica is full, for example on a single master, the RegistryStorage phase is `Failed` and a transition to air-gap waits;
- after two hours `D8RegistryStorageIncomplete` fires.

To fix it, mount the disk before the kubelet starts (in `/etc/fstab` without `nofail`) and delete the replica’s pod on that node. To keep the store on the node’s filesystem on purpose, delete the `.filesystem-before` file in the store’s directory on the node, then the replica’s pod:

```
d8 k -n d8-system delete pod -l app=registry-storage --field-selector spec.nodeName=<NODE>
```

A pull is failing on a node. Where do I look?

Start with the agent: it is on the path of every pull on the node. It is deployed as a static pod, so it is available even when the cluster is not.

The kubelet probes the agent on the address the container runtime uses, `127.0.0.1:5001`, and restarts it after a minute without an answer. An agent that stays not ready raises `D8RegistryAgentNotReady`, and one that keeps restarting raises `D8RegistryAgentRestartingTooOften`. Neither alert covers an agent that answers but cannot reach the API server: that one is ready.

To check the agent’s logs, use the command:

```
d8 k -n kube-system logs -l component=registry-agent --tail=100
```

To check the configuration the agent has, and whether it agrees with the cluster, use the command:

```
d8 k get registrynode <NODE> -o jsonpath='{.status}' | jq
```

The agent's metrics — its own view of the pulls passing through it. Prometheus collects them through the node exporter of the `monitoring-kubernetes` module: every 30 seconds the agent writes them to the node exporter's textfile directory, `/var/run/node-exporter-textfile/registry-agent.prom`. The `d8 registry_agent_*` series therefore come with `job="node-exporter"` and the node's name in the `node` label, for example:

```
sum by (node, kind, result) (rate(d8_registry_agent_requests_total[5m]))
```

Nothing scrapes the agent directly: it is a static pod precisely because it has to work when the API server does not, and a `kube-rbac-proxy` beside it would authenticate against that same API server. Without `monitoring-kubernetes` the series are not collected. On the node itself the same metrics, together with the agent's Go runtime metrics, are served on the loopback address:

```
ssh <NODE> 'curl -s http://127.0.0.1:4286/metrics | grep d8_registry_agent'
```

The configuration given to the container runtime. It is a single file, regardless of how many registries are configured:

```
ssh <NODE> 'cat /etc/containerd/registry.d/_default/hosts.toml'
```

If this file is missing, the agent has not applied a configuration yet: nothing on the node can pull, and the reason is in the agent's log. If the file is present and pulls still fail, the failure is beyond the agent — the metrics above name the target that failed and the error.

containerd can also pull past the agent, and this is by design. `_default/hosts.toml` sets no `server`, so containerd keeps the original registry as the last host and goes there directly when the agent does not answer, or answers `404` or another error status. This way images from public registries can still be pulled while the agent is down, but the direct attempt gets nothing the agent adds:

- no cluster credentials, only the pod's own pull secret, if it has one;
- no RegistryUpstream routes or mirrors, and no cache;
- no certificate authority of a private registry, so such a registry usually fails with `x509`;
- no in-cluster address: `registry.d8-system.svc:5001` does not resolve on the node, so platform images cannot be pulled this way at all.

Which error containerd reports when a pull fails depends on the step that failed and on the containerd version. While resolving the image reference, a tag or a digest:

- containerd 1.7 (`Containerd`) reports the first error it got other than `404`. If the agent did not answer or answered with an error status, that is the agent's error, even though the direct attempt was made after it: a refused connection or a TLS handshake timeout to the agent's address, or an error status such as `502 Bad Gateway`. The error of the direct attempt (`no such host`, `401`, `x509`) is reported only when the agent answered `404`, and then it hides that the agent did not find the image.
- containerd 2.2 (`ContainerdV2`) reports an error status other than `404` ahead of a `404`, and a `404` ahead of a failed connection; of two errors of the same kind, it reports the agent's. So an error status from the agent, such as `502 Bad Gateway`, is always what shows. A failed connection to the agent shows only when the direct attempt failed to connect too, and a failed connection of the direct attempt (`no such host`, `x509`) never shows. Otherwise the answer to the direct attempt shows: its error status, such as `403`, or `not found` for its `404`. An agent's `404` is reported as `not found` unless the registry answered the direct attempt with an error status.

Either way, check the agent's log and metrics first. When the direct attempt answers `404` too, as for a tag missing from a public registry, the pull is reported as `not found`. While fetching the manifest by digest, the image config and the layers, both versions report the agent's error whatever it was, `404` included: a layer the agent does not have is reported as `not found` even when the direct attempt then failed with another error.

While the agent does not answer, pulls are also slow to fail or succeed: on a test cluster with the agent frozen, pulls of platform images failed after about 20 seconds (a TLS handshake timeout to the agent), and images from public registries arrived over the direct path after about 50 seconds.

Modules from other ModuleSources: what the cache holds and what to do before air-gap

The cache holds every module the cluster runs, whichever ModuleSource it was installed from, and counts it towards completeness: a cluster does not go air-gap while a module it runs is missing from the store.

- A module whose ModuleSource is on the upstream's registry is filled from the upstream, like the platform itself.

- A module whose ModuleSource is on a registry of its own is filled from that registry, with the credentials and certificate authority the ModuleSource carries.

The platform's own modules — ModuleSource `deckhouse` — are stored under `system/deckhouse/modules`. Every other ModuleSource has a path of its own in the in-cluster registry, `system/deckhouse/module-sources/<ModuleSource>`: its modules under `<path>/<module>`, and its list of modules as the tags of `<path>`. A path apart, because the platform's is the upstream's as long as an upstream is configured, and a module another source ships under a name the platform also has would be replaced there by the upstream's build.

That path is the only place a node can pull the source's modules from once the upstream is gone, but the module's pods keep referring to their ModuleSource's registry until the source is changed. So before removing the upstream, point every such ModuleSource at its path, with the registry settings the platform's own source uses:

```
NAME=<ModuleSource>
REGISTRY=$(d8 k get modulesource deckhouse -o json | jq -c --arg name "$NAME" \
  '.spec.registry | .repo = "registry.d8-system.svc:5001/system/deckhouse/module-
sources/" + $name')
d8 k patch modulesource "$NAME" --type merge -p "{\"spec\":{\"registry\":$REGISTRY}}"
```

After that the module's images are rendered with the in-cluster address and are served from the cache.

What does destroying an air-gapped cloud cluster need from the store?

`dhctl destroy` of a cloud cluster whose provider is not built into the installer, such as Deckhouse Virtualization Platform, needs that provider's bundle: its schemas, OpenTofu modules and plugins. dhctl takes the bundle from its download directory (`--download-dir`, `/tmp/dhctl` by default) when it is there, and pulls it from the registry the cluster names otherwise. An air-gapped cluster names no upstream, so the bundle is pulled from the cluster's own store, through dhctl's SSH connection to a master. With the store empty or missing that image, destroy stops at `Get cluster configuration` with `MANIFEST_UNKNOWN`. The simplest way out is to run dhctl in a container that already has this bundle downloaded, such as the one a previous operation on this cluster ran in. Otherwise keep the store until the cluster is destroyed, or bring the bundle yourself: on a machine with access to the public registry, pull the digest the error names from the platform's image repository there, the one the cluster was installed from (for example, `registry.deckhouse.io/deckhouse/<edition>`; `system/deckhouse` in the error's address is only the store's name for that repository). A `d8 mirror` bundle of the release holds the image too. Unpack the image into the directory the error names.

How do I push a new Deckhouse Platform release or a new module version into an air-gapped cluster?

Once the upstream is removed from the configuration and the cluster has no internet access, `d8 mirror push` is the only way new images get into the in-cluster cache. The cache leader accepts the push, and the other replicas copy the images from it. As soon as a release is in the cache, the cluster sees it the same way it would see it in the upstream and updates by the usual rules: the release channel, the update windows, and manual approval if it is enabled.

You don't need to bring the upstream back for this, and you don't need to change `storage.source` either. The cache's completeness is checked against the image set the cluster needs, not against `expectedDigests`. Garbage collection keeps releases newer than the deployed one, so a pushed release stays in the store until it is deployed.

Step 1. Pull the images on a machine with internet access

Get the Deckhouse Platform edition and version running in the cluster:

```
d8 k -n d8-system get deployment deckhouse -o json | jq -r '.metadata.annotations |
{"core.deckhouse.io/edition","core.deckhouse.io/version"}'
```

Pull the Deckhouse Platform releases starting from the cluster's minor version. Minor versions are installed one after another, so the bundle has to contain every one in between, and `--since-version` pulls them:

```
d8 mirror pull --source='registry.deckhouse.io/deckhouse/<EDITION>' --
license='<LICENSE_KEY>' \
  --since-version='<X.Y>' ./d8-bundle
```

To update only a module without pulling the platform, give its current version. The command pulls that version, the newer ones, and the versions on the release channels:

```
d8 mirror pull --source='registry.deckhouse.io/deckhouse/<EDITION>' --
license='<LICENSE_KEY>' \
  --no-platform --include-module='<MODULE>@<VERSION>' ./d8-bundle
```

The Updating DP in air-gapped environment guide explains how to make the bundle smaller and how to build the list of modules.

Move the bundle to the machine you will run `d8 mirror push` from. Make sure the cache's store has room for it: a push into a full store gets `507 Insufficient Storage` (see What happens when the cache's store is full?).

Step 2. Check the publication endpoint

Images get into the cache only through the publication endpoint: the `registry-push` Ingress in the `d8-system` namespace, which passes the push on to the cache leader. A push sent straight to a master node, around the Ingress, is refused: the port that accepts writes there takes them only from the ingress controller and from the cache itself (see What protects the ports on control-plane nodes).

Check whether the cluster has the `registry-push` Ingress and whether an ingress controller serves it:

```
d8 k -n d8-system get ingress registry-push -o jsonpath='{.spec.ingressClassName}{"\t"}
{.spec.rules[0].host}{"\n"}'
d8 k get ingressclass
```

- The Ingress exists and its class is among the IngressClasses: go on to step 3.
- The Ingress exists, but its class is not among the IngressClasses: set the `ingressClass` parameter of the `registry` ModuleConfig to the class of an existing controller, then go on to step 3.
- There is no Ingress: it is created only when `publicDomainTemplate` is set and the `cert-manager` module is enabled. Set the parameter and enable the module: the Ingress appears on the next run of the `registry` module. If the cluster cannot have the Ingress, set `primary.upstream` to a registry the cluster can reach that holds the images, let the cache fill from it, and remove the upstream again.

Run the `d8 k` commands in the next steps where you have access to the cluster's API. If the bundle is on another machine, carry the values over to it.

Step 3. Push the images

```
HOST=$(d8 k -n d8-system get ingress registry-push -o jsonpath='{.spec.rules[0].host}')
PUSH_SECRET=$(d8 k -n d8-system get secret registry-storage-push -o json)
d8 mirror push ./d8-bundle "$HOST/system/deckhouse" \
  --registry-login="$(echo "$PUSH_SECRET" | jq -r .data.username | base64 -d)" \
  --registry-password="$(echo "$PUSH_SECRET" | jq -r .data.password | base64 -d)"
```

What the push needs:

- The name in `$HOST` resolves to the ingress controller's address on the machine you push from.
- If `whitelistSourceRanges` is set, that machine's address is in the list.
- The machine trusts the Ingress certificate. Let's Encrypt can't issue one for an air-gapped cluster, so such clusters usually use their own ClusterIssuer or the `CustomCertificate` mode (the `https` parameter). If the machine doesn't trust the certificate authority, pass its certificate in the `SSL_CERT_FILE` environment variable.
- The cache has a leader. While there is none, the push is refused; look at the replicas in the RegistryStorage status (see How do I check the state of the in-cluster cache?). If the leader changed during the push, for example because a master node rebooted, run the command again.

Do not push while a garbage collection is running: check `d8 k get registryoperation` first. A push that overlaps a collection can fail, and it can also report success and leave an image incomplete. If a push overlapped a collection, run it again after the collection finishes (see When it runs, and what happens during a run?).

Step 4. Check that the release has reached the cluster

Check the state of the cache. The other replicas copy the new release from the leader before it is deployed:

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq
'{phase,fill,leader,allReplicasFull}'
```

Check that the cluster sees the new Deckhouse Platform release or module release:

```
d8 k get deckhousereleases
d8 k get modulereleases
```

From here on the release is deployed according to the Deckhouse Platform update settings and the module update settings.

Image availability alerts from `extended-monitoring` for the in-cluster registry

With the module in charge, every image reference in the cluster names `registry.d8-system.svc:5001`. Nodes pull by that address through the node agent on the loopback, not through cluster DNS. The image availability exporter of the `extended-monitoring` module checks images from a pod, where the name is looked up in cluster DNS instead, so it reports every such image as `… ImageAvailabilityUnknownError`:

- without the in-cluster cache, or while no storage replica is ready, the name does not resolve from a pod: `no such host`;
- with the cache, the name is a headless Service that resolves to the master nodes running ready storage replicas. The storage answers there with a certificate issued by the module's own certificate authority, which the exporter does not trust: `x509: certificate signed by unknown authority`.

Nothing is wrong with the images: what the exporter would detect is covered by this module's own alerts — `D8RegistryStorageIncomplete`, `D8RegistryUpstreamProbeFailing`, `D8RegistryNodeNotConverged`. Exclude the in-cluster address from the exporter's checks:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: extended-monitoring
spec:
  version: 2
  settings:
    imageAvailability:
      ignoredImages:
      - 'registry\.d8-system\.svc:5001/.*'
```

Each entry is a regular expression. Images from other registries are still checked.

How do I check the state of the in-cluster cache?

The state of the cache is published in the status of the RegistryStorage resource. To view it, use the command:

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq
```

Description of the response fields:

- `replicas` is the only place where completeness information is reported, and each entry is the replica's own account of itself. A replica reporting `full: true` alongside an `error` is not complete: `full` says what it holds, and the error says whether its last pass finished, or whether its registry or disk has stopped working (see `leader`). `syncerStartedAt` is when the syncer that wrote the entry started, set once that syncer has read its own store; the upstream is dropped only when the leader's entry carries the start of the syncer running in its pod now.
- `leader` is the replica that fills from the upstream and serves as the replication source for the others. The election is deliberately not symmetric: once a replica holds the whole expected set, a leader that cannot complete it by itself steps aside for that replica. A leader cannot complete the set by itself when there is no upstream, when its last pass failed, when its store refuses writes, or when it no longer renews its lease. This is what keeps an air-gapped cluster from deadlocking with an empty leader and a full follower. A leader that is still completing the set from the upstream

keeps leading: a follower copies from it and may finish first, and moving the lease then would gain nothing. A replica on a node unreachable for about two minutes is not counted in the election, and a replica whose registry stops answering or whose disk stops accepting writes reports it in `healthError` and `error` and steps aside for a full replica, if there is one. A leader whose store has lost the set (the deployed release or most of a sample of the set is missing on its disk, or `totalDigests` is at most half of its `verifiedDigests`) steps aside for a replica full for the current set, reporting `full: false` and `verifiedDigests: 0`, and copies the set from it. With no such replica it keeps leading, reports `full: false` and `verifiedDigests: 0` all the same, and fills the set again from the upstream (in an air-gapped cluster, it counts its store again, and the set comes back only with a new `d8 mirror push`).

Which images are in the cache?

Every image of the cache, by the digest of its manifest, is listed by the RegistryImage resource of the `registry.deckhouse.io` group. It is not stored anywhere: each request asks every replica of the cache, and each replica answers from an index it keeps in memory and rebuilds by reading its store every two minutes and after every operation. The answer is always up to date, and building it puts no load on the disks of the master nodes.

Request examples:

```
d8 k get registryimages
d8 k get registryimages -o wide
d8 k get registryimages -l registry.deckhouse.io/module=prometheus
d8 k get registryimages -l registry.deckhouse.io/state=Corrupted
d8 k get registryimages -l registry.deckhouse.io/in-set=false
d8 k get registryimage sha256-<digest> -o yaml
d8 k explain registryimages
```

Image fields in the response:

- `owners` — every place the image is used: the module and the image name from `images_digests.json`, and the Deckhouse Platform releases or module versions that declare it. One digest can be used by several modules; all of them are listed.
- `inSet` — the cluster release or module that uses the image. An image outside the set or left by an older release may be listed and is later removed by the garbage collection. Such images cannot be removed by hand.
- `state` and `replicas[].state` — where the image is on each replica:
  - `Present` — the replica holds the manifest and every blob of the image;
  - `Missing` — an image of the set the replica does not hold at all;
  - `Incomplete` — the replica holds the manifest without some of its blobs, which can happen when a pull-through cache served only the manifest;
  - `Corrupted` — one of the image’s blobs does not match its digest;
  - `Unknown` — the replica did not answer, or has not read its store since it started. When no replica can answer, the request fails with `ServiceUnavailable` and names each replica’s reason.
- `size` — the full size of the image: its manifest and every layer, each counted once.
- `reclaimableSize` — the amount of space freed on a replica if only this image is removed.

An image is made of layers, and a layer that several images share is stored once. So the sizes of all images add up to more than the cache takes on disk, and removing one image frees only the layers no other image uses: `reclaimableSize`. The images of the platform share few layers, so `reclaimableSize` and `size` are usually close.

The counts by state are in the RegistryStorage status, under `replicas[].images`, and stay there while registry-controller is not running. During this time, the `registry.deckhouse.io` group is not served: `kubectl api-resources` reports an error for it, and namespaces being deleted stay `Terminating` with the `NamespaceDeletionDiscoveryFailure` condition until a replica of the controller answers again. The `D8RegistryControllerDown` alert fires at the same time.

Reading RegistryImage requires the `get` and `list` permissions on `registryimages` of the `registry.deckhouse.io` group. The user must have the `registry.view` RBACv2 capability.

Which images take the most space?

Sort the list by `size` or `reclaimableSize`. kubectl sorts in ascending order, so the largest images are at the end of the output, and images without a size, such as `Missing` ones, are at the start. Sorting works together with any label selector.

```
# The 20 largest images.
d8 k get registryimages --sort-by=.size | tail -n 20
# The 20 images that take the most space on their own: layers no other image uses.
# With -o wide both sizes are shown.
d8 k get registryimages --sort-by=.reclaimableSize -o wide | tail -n 20
# The largest images outside the set, which the garbage collection removes.
d8 k get registryimages -l registry.deckhouse.io/in-set=false --sort-by=.reclaimableSize -o wide |
tail -n 20
# The images of one module by size.
d8 k get registryimages -l registry.deckhouse.io/module=prometheus --sort-by=.size
```

How do I check and repair images in the cache?

Every replica reads its store once an hour in the background and compares every blob with its digest. The read speed is about 4 MiB/s on a store of 15 GB, with a maximum of 16 MiB/s. The kernel drops what it read from the page cache so that etcd and the API server on the same master do not react to the load. Pulls are not blocked. The time when the last full pass finished is in `RegistryStorage.status.replicas[].scrubbedAt`.

A blob that does not match its digest is recorded in `RegistryStorage.status.replicas[].corruptedBlobs`, and the images that use it are marked `Corrupted`. The following happens automatically:

- While an image of the set uses the broken blob, the replica is not `full`. A leader then hands the lease to a replica whose set has no broken blobs, and the nodes pull from it.
- The module creates a RegistryOperation of type `Repair` for the broken blobs as soon as no other operation runs. `Repair` downloads a whole copy of each broken blob from another replica or from the upstream (if available), checks it against its digest and only then replaces the broken copy. A blob for which no whole copy is found stays as it was.
- One `Repair` takes up to 50 blobs. If it did not fix everything, `Repair` is repeated four hours later; if there are more than 50 broken blobs, it starts with the next batch. After two hours the `D8RegistryStoreCorrupted` alert fires.

To check the store manually, create a RegistryOperation with the `Verify` operation type:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: verify-
spec:
  type: Verify
  # Optional: only these images or blobs, and only these replicas.
  verify:
    digests: ["sha256:<digest>"]
    nodes: ["<master-node>"]
```

The read speed reaches 32 MiB/s. A store of 15 GB is checked in about eight minutes per replica, one replica at a time.

To repair images manually, for example after pushing a release again with `d8 mirror push`, create a RegistryOperation with the `Repair` operation type with their digests (up to 50):

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryOperation
metadata:
  generateName: repair-
spec:
  type: Repair
  repair:
    digests: ["sha256:<digest>"]
```

`Repair` of a whole image reports `Intact` when every blob matches its digest. The outcome for each digest is in `status.replicas[].repair.digests` of the operation. Only one operation runs at a time. If an operation was created while a garbage collection runs, it fails with `AnotherOperationActive` and has to be created again.

Creating operations needs the RBACv2 capability `registry.edit`.

How is the cache updated?

A new Deckhouse Platform release can change the images of the cache itself, the `registry-storage` StatefulSet in `d8-system`. The StatefulSet uses the `OnDelete` update strategy, and its replicas are replaced by `registry-controller`: one at a time, followers first and the leader last, and only while every other replica is ready. A replica on the old revision that has never started or is in `CrashLoopBackOff` is replaced at once.

A replaced replica pulls its new images after it is deleted, from the cache it was part of. So the controller replaces a replica only while something else can serve them:

- a follower: the leader keeps serving;
- the leader: once a follower holds the whole image set. The lease then goes to whichever replica holding the whole set takes it first: a follower holding it or the leader's own replacement, which comes back on the same node over the same store;
- any replica: the upstream registry, if the cluster has one.

One master node in an air-gapped cluster

With a single replica and no upstream, nothing else can serve the images. The new images therefore have to be on the master before the replica is replaced. The module keeps them there with the `registry-storage-image-holder` DaemonSet: on every master it runs each image of the cache's pod as `/pause`. When the images change, the holder pulls the new ones from the running cache, and as long as it runs them the kubelet does not delete them from the node. Once the holder on the master runs every new image, the controller replaces the replica, and the new pod starts from the images already on the node.

While the only replica of the cache restarts, the nodes cannot pull images from it. How long this takes depends on stopping the old replica and starting the new one. The kubelet retries failed pulls with a back-off of up to 5 minutes, so some pods may start with an extra delay after the cache is back.

The holder requests 10m CPU and 10Mi of memory per image on each master, 40m and 40Mi in total. It holds the images on every master, but the controller only uses it when the cache has one replica and no upstream.

Several master nodes in an air-gapped cluster

The leader is replaced last, and only once a follower holds the whole image set. Otherwise the lease would move to a follower with images missing, and the images only the old leader had would be gone from the cluster. The image holder does not replace that check. A follower counts only if it was found full against the same set as the leader: after an update that changes the Deckhouse Platform release or the modules, a replaced follower still reports the set it held before until it has copied the new one. Normally this means waiting while a follower copies the set from the leader. If the leader itself does not hold the whole set, the followers cannot complete it either: they copy from the leader. Then the update continues only once the Deckhouse Platform release and the modules the cluster uses are pushed into the cache.

What the UpdateBlocked event means

While the controller waits, it writes an `UpdateBlocked` event to the StatefulSet with the replica, the reason and what ends the wait. It writes the event again when what the wait depends on changes, and otherwise every five minutes while the wait lasts:

```
d8 k -n d8-system get events --field-selector involvedObject.name=registry-
storage,reason=UpdateBlocked
```

The same reason is the `reason` label of the `d8_registry_storage_update_blocked` metric, and the `D8RegistryStorageUpdateBlocked` alert fires if the wait lasts longer than an hour:

- `HolderNotReady` — one replica, and the holder on its master does not run every new image yet. If its pod is in `ImagePullBackOff`, the cache does not hold an image: push the Deckhouse Platform release the cluster is updating to with `d8 mirror push`, and the holder pulls it on its next retry. If its pod is in `CrashLoopBackOff` or `CreateContainerError`, the holder has the image but cannot run it, and a push does not help: the cause is in the events of the pod. Otherwise the holder is still rolling or pulling, and nothing needs to be done.
- `SingleReplicaWithoutHolder` — one replica, and the holder DaemonSet has no pod on its master. For a moment this is normal while the DaemonSet creates its pod on the node. If it lasts, find out why the DaemonSet does not schedule a pod there: `d8 k -n d8-system describe`

`daemonset registry-storage-image-holder`.

- `LeaderWithoutFullFollower` — only the leader remains to be replaced, but no follower holds the whole set. If the leader holds it, wait for a follower to finish copying. If the leader reports that it does not hold it, pull the Deckhouse Platform release and the modules the cluster uses with `d8 mirror pull`, as described in the air-gapped update guide, and push them into the cache with `d8 mirror push`, as in step 2 of Going air-gapped. If the leader has not reported yet, reports an error or cannot tell which set the cluster needs, a push does not help, and the update continues only after a pass of the leader succeeds: look at the log of the `syncer` container of the leader's pod.
- `LeaderNotServing` — the storage lease names a master without a serving replica. It moves on its own once the lease expires.
- `StorageUnknown` — the `RegistryStorage` resource does not exist yet. The controller creates it. If it does not appear, check the log of `registry-controller`.

To see what the holders and the replicas are doing:

```
d8 k -n d8-system get pods -l app=registry-storage-image-holder -o wide
d8 k get registrystorage registry -o jsonpath='{.status.replicas}' | jq
```

Who can read from and write to the in-cluster cache?

The cache's registry accepts three accounts. Each of them is limited to the platform's path, `system/deckhouse`, and everything under it: that is where the cache keeps the platform, its modules, and the modules of other ModuleSources. A request for any other repository is refused, whatever the account.

Account	Who uses it	What it may do
`registry-ro`	the node agent on every node	pull; list the store's repositories
`registry-sync`	the cache's own pods only	pull, push and delete: filling from the upstream, replication, garbage collection
`registry-rw`	`d8 mirror push`, through the publication endpoint	pull and push

- The read account is held by every node, so it can change nothing the nodes pull.
- The sync account is the one that writes into the store from inside the cluster. Its password is used only by the cache's pods: the registry controller and `registry-packages-proxy` read the Secret `registry-storage-client`, which carries the certificate authority, the read account and the replicas' addresses and nothing else.
- The publication account is for the platform administrator. It is how a new release reaches an air-gapped cluster, so it can replace the platform's images by design: whoever can read the Secret `registry-storage-push` in `d8-system` can replace what every node pulls. Grant read access to that Secret only to those who publish the platform. The account can neither delete images nor list the store's repositories: images are removed by the garbage collection.

Push to the platform's path, `<publication endpoint>/system/deckhouse`, as in the air-gap example; a push to any other path is refused.

The v1 implementation

The information below applies to a cluster still running the v1 implementation of the module (DP up to and including 1.77), configured through the `deckhouse` ModuleConfig.

How to Migrate to the registry module?

This procedure moves a cluster from the deprecated format (without the `registry` module) to the v1 implementation. It is not a step of the migration to the current implementation: a cluster in the deprecated format needs no preparation for the upgrade to DP 1.78. See how the migration works.

During the migration, Containerd v1 will switch to the new registry configuration format. Containerd v2 uses the new format by default. For more details, see the section with a description of configuration methods.

For containerd v2

1. Switch to using the `registry` module. To do this, specify the `Unmanaged` mode parameters in the ModuleConfig `deckhouse`. If you are using a registry other than `registry.deckhouse.io`, refer

to the `deckhouse` module documentation for proper configuration.

You can view the current registry settings using the following command:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller global values | yq e '.modulesImages.registry' -
```

Specify this configuration when setting up the `Unmanaged` mode:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Replace with your license key
```

2. Wait for the switch to complete. Example status output:

```
conditions:
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

For Containerd v1

✖
- During the switch, containerd v1 will be restarted.
- During the switch, containerd v1 will be migrated to the new registry configuration scheme.
- During the switch, custom registry configurations for containerd v1 will be temporarily unavailable.

1. Make sure that nodes with containerd v1 do not have any custom registry configurations located in the `/etc/containerd/conf.d` directory.
2. If configurations are present, you need to migrate to the new registry configuration format in containerd. To do this, add new configuration files to the `/etc/containerd/registry.d` directory. These configurations will take effect after switching to the `registry` module. To add configurations, prepare a NodeGroupConfiguration. For more details, see the section with a description of configuration methods. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # The step can be arbitrary, as restarting the containerd service is not
required
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - <<
EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        [host."https://${REGISTRY_URL}".auth]
          username = "username"
          password = "password"
    EOF
```

3. Apply the NodeGroupConfiguration. Wait until the configuration files appear in the `/etc/containerd/registry.d` directory on all nodes.
4. Verify that the configurations are working correctly. To do this, use the following command:

```
# For HTTPS:
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/registry/path:tag

# For HTTP:
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/registry/path:tag
```

5. Switch to using the `registry` module. To do this, specify the `Unmanaged` mode parameters in the ModuleConfig `deckhouse`. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for proper configuration.

   You can view the current registry settings using the following command:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller global
   values | yq e '.modulesImages.registry' -
   ```

   Specify this configuration when setting up the `Unmanaged` mode:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Replace with your license key
```

6. After applying, wait for the following message to appear in the switch status:

   Example output:

   ```
   conditions:
   # ...
   - lastTransitionTime: "2025-08-13T15:22:34Z"
     message: |
       Check current nodes configuration
       2/2 node(s) Unready:
       - master-0: has custom toml merge containerd configuration
       - worker-5e389be0-578df-s5sm5: has custom toml merge containerd
   configuration
     reason: Processing
     status: "False"
     type: ContainerdConfigPreflightReady
   ```

   This message means that there are old registry configurations on the nodes located in the `/etc/containerd/conf.d` directory. The switch to the new containerd configuration is currently blocked. To allow the switch, you need to remove the old configuration files.

7. Remove the old configuration files to allow switching to the `registry` module. To do this, create a NodeGroupConfiguration. Example of a NodeGroupConfiguration manifest:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth-delete.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    file="/etc/containerd/conf.d/old-config.toml"

    [ -f "$file" ] && rm -f "$file"
```

8. After removing the old configurations, make sure that the switch has resumed. Example of the switch status:

```
conditions:
# ...
- lastTransitionTime: "2025-08-13T16:42:09Z"
  message: ""
  reason: ""
  status: "True"
  type: ContainerdConfigPreflightReady
```

9. Wait for the switch to complete. Example of the switch status:

```
conditions:
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

10. Delete the NodeGroupConfiguration created in the step for deleting old configuration files:

```
d8 k delete nodegroupconfiguration containerd-additional-config-auth-delete.sh
```

To verify that NodeGroupConfiguration has been deleted, use the command:

```
d8 k get nodegroupconfiguration
```

The list should not contain the NodeGroupConfiguration to be deleted (for this example, `containerd-additional-config-auth-delete.sh`).

How to Migrate Back from the Registry Module?

✖
- This is a deprecated registry management format.
- During the switch, containerd v1 will be restarted.
- During the switch, containerd v1 will be migrated to the legacy registry configuration scheme.
- During the switch, custom registry configurations for containerd v1 will be temporarily unavailable.

1. Switch the registry to `Unmanaged` mode. If you are using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for proper configuration.

   Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
         unmanaged:
           imagesRepo: registry.deckhouse.io/deckhouse/ee
           scheme: HTTPS
           license: <LICENSE_KEY>
   ```

2. Check the switch status using the instruction. Example output:

   ```
   conditions:
   # ...
   - lastTransitionTime: "..."
     message: ""
     reason: ""
     status: "True"
     type: Ready
   hash: ..
   mode: Unmanaged
   target_mode: Unmanaged
   ```

3. Switch the registry to the non-configurable `Unmanaged` mode. Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: deckhouse
   spec:
     version: 1
     enabled: true
     settings:
       registry:
         mode: Unmanaged
   ```

4. Check the switch status using the instruction. Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

5. If containerd v1 is used and custom registry configurations are applied in the cluster, they must be replaced with the old format. To do this, prepare the registry configurations in the old format. These configurations do not need to be applied at this stage. Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # To add a file before the '032_configure_containerd.sh' step
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
            endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
            username = "username"
            password = "password"
            # OR
            auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

6. Delete the `registry-bashible-config` secret. This will trigger containerd v1 to switch back to

the legacy registry format:

```
d8 k -n d8-system delete secret registry-bashible-config
```

7. After deletion, wait for the switch to complete. Use the instruction to track the progress. Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

8. If containerd v1 is used, apply the previously prepared NodeGroupConfiguration with custom registry configurations.
9. Disable the `registry` module. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: false
  settings: {}
  version: 1
```

How to check the registry mode switch status?

The status of the registry mode switch can be retrieved using the following command:

```
d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
map_values(@base64d) | .conditions = (.conditions | from_yaml)'
```

Example output:

```
conditions:
  - lastTransitionTime: "2025-07-15T12:52:46Z"
    message: 'registry.deckhouse.io: all 157 items are checked'
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
  - lastTransitionTime: "2025-07-11T11:59:03Z"
    message: ""
    reason: ""
    status: "True"
    type: ContainerdConfigPreflightReady
  - lastTransitionTime: "2025-07-15T12:47:47Z"
    message: ""
    reason: ""
    status: "True"
    type: TransitionContainerdConfigReady
  - lastTransitionTime: "2025-07-15T12:52:48Z"
    message: ""
    reason: ""
    status: "True"
    type: InClusterProxyReady
  - lastTransitionTime: "2025-07-15T12:54:53Z"
    message: ""
    reason: ""
    status: "True"
    type: DeckhouseRegistrySwitchReady
  - lastTransitionTime: "2025-07-15T12:55:48Z"
    message: ""
    reason: ""
    status: "True"
    type: FinalContainerdConfigReady
  - lastTransitionTime: "2025-07-15T12:55:48Z"
    message: ""
    reason: ""
    status: "True"
    type: Ready
mode: Direct
target_mode: Direct
```

The output displays the status of the switch process. Each condition can have a status of `True` or `False`, and may contain a `message` field with additional details.

Description of conditions:

Condition	Description
`ContainerdConfigPreflightReady`	State of the containerd configuration preflight check. Verifies there are no custom containerd auth configurations on the nodes.
`TransitionContainerdConfigReady`	State of preparing the containerd configuration for the new mode. Verifies that the configuration contains both the old and new mode settings.
`FinalContainerdConfigReady`	State of finalizing the switch to the new containerd mode. Verifies that the containerd configuration has been successfully applied and contains only the new mode settings.
`DeckhouseRegistrySwitchReady`	State of switching Deckhouse and its components to use the new registry. `True` means Deckhouse successfully switched and is ready to operate.

Condition	Description
`InClusterProxyReady`	State of In-Cluster Proxy readiness. Checks that the In-Cluster Proxy has started successfully and is running.
`CleanupInClusterProxy`	State of cleaning up the In-Cluster Proxy if it is not needed in the selected mode. Verifies that all related resources have been removed.
`NodeServicesReady`	State of Node Services Manager and Static-Pod registry readiness. Verifies that the Node Services Manager is successfully launched and operational, and that the Static-Pod registry has been successfully deployed by it.
`CleanupNodeServices`	State of cleaning up the Node Services Manager and Static-Pod registry if they are not needed in the selected mode. Verifies that all related resources have been removed.
`RegistryContainsRequiredImages`	State of checking the registry for the presence of required images.
`Ready`	Overall state of registry readiness in the selected mode. Indicates that all other conditions are met and the `module` is ready to operate.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module registry: usage examples

⚠ The module lifecycle stage: Preview

⚠ If, during the switching process, the image of a module did not reload and the module did not reinstall, use the instructions to resolve the issue.

Enabling the module

To have the module manage how the cluster pulls images, set `mode: Managed` and specify the registry to pull from:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  version: 1
  enabled: true
  settings:
    mode: Managed
    primary:
      upstream:
        host: registry.deckhouse.io
        path: /deckhouse/ee
        scheme: HTTPS
        auth:
          license: <LICENSE_KEY>
```

The module publishes a ready-made version of this for your cluster, with the address, path, scheme, certificate authority and credentials of the registry it is already pulling from:

```
d8 k -n d8-system get secret registry-suggested-config -o jsonpath='{.data.registry-mc\.yaml}'
| base64 -d
```

Review it, then apply it. It saves a transcription rather than typing: those values are spread across a secret and a docker configuration blob, and retyping them by hand is where a truncated path or the wrong authority comes from — on the one setting that decides whether the cluster can pull at all.

To see the change take effect, use the following commands:

```
d8 k get registryconfig registry -o jsonpath='{.status}' | jq
d8 k get registrynodes -o custom-columns=\
NODE:.metadata.name,APPLIED:.status.observedGeneration,OK:.status.reconciled,BACKENDS:.status.activeBackends
```

Turning the in-cluster cache on

To turn on the in-cluster cache, add the module's `storage.cache` ModuleConfig setting and specify a size for the store:

```
spec:
  settings:
    mode: Managed
    primary:
      upstream:
        host: registry.deckhouse.io
        path: /deckhouse/ee
        auth:
          license: <LICENSE_KEY>
    storage:
      cache: true
      size: 50Gi
```

Nothing on any node is reconfigured. The container runtime already asks the node agent about every registry, and the agent starts preferring the cache with the upstream as a fallback — so a cache miss is a slower pull rather than a failed one, from the first moment.

To check how full the in-cluster cache is, use the following command:

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq
'{phase,fill,leader,allReplicasFull}'
```

Turning the in-cluster cache off again is the same change in reverse, and just as safe. The blobs on disk

are left alone, so turning it on again refills from what is already there rather than from scratch — see how to reclaim that space if you do not intend to.

Going air-gapped

An air-gapped cluster has no upstream (not set in the `primary.upstream` parameter). For such a cluster, the cache is the only source of images, and `d8 mirror push` is the way in. Because completeness has to be decidable before the cache can be trusted alone, `storage.source` must describe the expected image set.

To move a cluster to air-gap, follow these steps:

1. Pull the images on a machine that has internet access:

   ```
   d8 mirror pull --license <LICENSE_KEY> ./d8-bundle
   ```

2. Push them into the cluster, through the publication endpoint:

   ```
   PUSH_SECRET=$(d8 k -n d8-system get secret registry-storage-push -o json)
   d8 mirror push ./d8-bundle registry.example.com/system/deckhouse \
     --registry-login "$(echo "$PUSH_SECRET" | jq -r .data.username | base64 -d)" \
     --registry-password "$(echo "$PUSH_SECRET" | jq -r .data.password | base64 -d)"
   ```

   The publication endpoint exists only while the cache is enabled, `publicDomainTemplate` is set and the `cert-manager` module is enabled; see Air-gapped clusters.

3. Describe the expected set of images in the cache, and remove the upstream:

   ```
   spec:
     settings:
       mode: Managed
       storage:
         cache: true
         size: 50Gi
         source:
           bundleRef: d8-mirror-bundle
           expectedDigests: 459
   ```

The upstream is not removed from the nodes the moment you remove it from the configuration. It is removed once the cache leader holds the whole expected set and nothing in the platform namespaces (`kube-system` and `d8-*`) still references the upstream — this is the one transition that could otherwise leave every node with nowhere to pull from, so it waits, and says so. When the upstream is removed, the registry restarts, and the RegistryStorage resource may briefly report `Failed`: this is expected, wait for it to return to `Ready`.

> ⚠ Workloads in other namespaces are not counted. If the images of your own workloads reference the upstream address, move them onto the in-cluster address before removing the upstream: once it is removed, they have nowhere to pull from.

To check the status of the transition, use the following commands:

Check whether the cache leader holds the whole expected set of images, and how many platform workloads still name the upstream (`upstreamReferences`):

```
d8 k get registrystorage registry -o jsonpath='{.status}' | jq
'{safeToDropUpstream,fill,upstreamReferences}'
```

A nonzero `upstreamReferences` means that a rollout of platform workloads onto the in-cluster address has not finished. The `registry-controller` log names up to five of the referencing objects each time the count changes. To see the last such line, run:

```
d8 k -n d8-system logs -l app=registry-controller -c registry-controller --prefix --tail=-1 | grep
'still names the upstream' | tail -1
```

Get the value of `effectiveUpstream`:

```
d8 k get registryconfig registry -o jsonpath='{.status.effectiveUpstream}' | jq
```

While `effectiveUpstream` is still set, the cluster is using it. When it becomes empty, the cluster is air-gapped.

If the transition does not finish, find what holds it:

- The cache is not complete (`safeToDropUpstream: false` with the leader not `full`): wait for the fill, or check the `error` of the leader in `.status.replicas`.
- `upstreamReferences` stays above zero: finish or fix the rollout of the objects named in the `registry-controller` log.
- The leader is `full` and `upstreamReferences` is `0`: the upstream is held until the `syncer` running on the leader's node has read its store and stamped its report with `syncerStartedAt`, normally within two minutes of its start. This does not end while the leader's storage pod is not Ready, or while that pod runs a `syncer` from before this field existed (during a module update). Deleting the leader's storage pod is safe while the upstream is held: the new `syncer` reads its store and stamps its report, normally within two minutes. To see the leader's report and the reason the controller gives:

  ```
  d8 k get registrystorage registry -o json | jq '.status as $s | $s.replicas[] |
  select(.node == $s.leader) | {node,full,error,syncerStartedAt}'
  d8 k -n d8-system logs -l app=registry-controller -c registry-controller --prefix --tail=-1 |
  grep 'the upstream is held' | tail -1
  ```

To stay on the upstream instead, put the same `primary.upstream` back into the ModuleConfig: the cluster has been using it all along and keeps doing so.

Adding another registry

A registry that is not the source of DP component images is declared as its own resource — RegistryUpstream — rather than another field in the ModuleConfig. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryUpstream
metadata:
  name: virtualization-images
spec:
  match: images.virtualization.example.com
  upstream:
    host: vendor.example.com
    path: /virtualization
    scheme: HTTPS
    auth:
      username: robot
      password: <PASSWORD>
```

Pulls naming `images.virtualization.example.com` are then routed to `vendor.example.com/virtualization` by the agent on every node, with the credentials and certificate authority held by the cluster rather than by each workload. Nothing on any node is reconfigured for this.

After creating the RegistryUpstream, check that it has been accepted. The object is rejected in the following cases:

- It conflicts with the primary registry or another object claiming the same `match` value. The object is rejected rather than merged.
- It contains an invalid field, such as a `ca` that is not a PEM certificate.
- Adding the object exceeds either of the following limits:
  - Settings from no more than 256 RegistryUpstream objects can be applied at a time.
  - The combined size of the accepted objects' `spec` fields in JSON format must not exceed 256 KiB.

To check whether the object has been accepted, run the following command:

```
d8 k get registryupstreams -o custom-columns=\
NAME:.metadata.name,MATCH:.spec.match,ACCEPTED:.status.conditions[0].status,REASON:.status.conditions[0].rea
son
```

Pulling from a mirror near the nodes

Nodes far from the control plane can pull the image set from a registry near them. Prepare the registry first: it has to serve the image set by digest under one prefix. For example:

- a Harbor proxy-cache project of `registry.deckhouse.io`: the prefix is `/<project>/deckhouse/<edition>`;
- a project filled with `d8 mirror push <bundle> harbor.example.com/<project>/<path>`: the prefix is `/<project>/<path>`.

Then select the nodes that ask it first:

```
apiVersion: deckhouse.io/v1alpha1
kind: RegistryMirror
metadata:
  name: gpu-eu
spec:
  nodeSelector:
    matchLabels:
      node.deckhouse.io/group: gpu-eu
  source:
    host: harbor.eu.example.com
    path: /d8-proxy/deckhouse/ee
    auth:
      username: robot
      password: <PASSWORD>
```

The agent on every node of the `gpu-eu` group now asks `harbor.eu.example.com` for content named by digest before the cache or the upstream. Tags are still resolved by the cache or the upstream, and whatever the mirror does not serve is pulled from there.

For a public project, leave out `auth`: the agent asks the mirror anonymously, with an anonymous token where the registry wants one.

Check that the mirror has been accepted and for how many nodes:

```
d8 k get registrymirrors -o wide
```

The object is rejected when a node it selects is served by a mirror created earlier (the reason is `NodeConflict`; narrow the `nodeSelector` of one of them), or when a field of `spec` is invalid (`InvalidSpec`).

How much the mirror serves is visible in the `d8 registry_agent mirror requests_total` metric of each node, by result: `hit`, `miss`, `refused`, `unavailable` and `digest_mismatch`. Many misses mean the mirror does not hold what the nodes pull; `refused` usually means wrong credentials.

Pulling from a private registry without declaring it

Nothing needs to be declared to pull from a registry the module does not know about. When the module manages how images are pulled, the node agent is what the container runtime turns to for any registry, including ones the module has no information about. An unconfigured registry is proxied untouched, along with whatever credentials the pull request already carried.

So an ordinary `imagePullSecret` behaves exactly as it would on a cluster where this module was never enabled.

To create a secret with the credentials of the private registry, run:

```
d8 k create secret docker-registry my-private-registry \
  --docker-server=private.example.com \
  --docker-username=robot \
  --docker-password=<PASSWORD>
```

To use this secret when pulling images, add it to the pod's `imagePullSecrets`:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  imagePullSecrets:
  - name: my-private-registry
  containers:
  - name: app
    image: private.example.com/team/app:v1
```

It is worth creating a `RegistryUpstream` only if you want the cluster to hold the credentials instead of every workload doing so individually, or if the registry needs a certificate authority the nodes do not have.

Turning the module off again

To have the module stop managing how images are pulled, set the `mode` parameter to `Unmanaged`:

```
spec:
  settings:
    mode: Unmanaged
```

The module then manages nothing: its components are removed, the node configuration it wrote is withdrawn, and the cluster goes back to pulling from the registry recorded in the `deckhouse-registry` secret — which is where it was pulling from before the module was ever enabled.

Cache data on the control-plane nodes is deliberately left behind: turning the cache on again fills in only the missing images from the registry, rather than starting from scratch. If you do not plan to use the cache again, see how to reclaim that space.

Examples for the v1 implementation

Everything below applies to a cluster still running the v1 implementation (DP up to and including 1.77), configured through the `deckhouse` ModuleConfig. See how the migration works.

Switching to the `Unmanaged` Mode

To switch an already running cluster to `Unmanaged` mode, follow these steps:

✖ Changing the registry in `Unmanaged` mode will result in a full restart of all DP components.

1. Make sure the `registry` module is enabled and running. To do this, execute the following command:

   ```
   d8 k get module registry -o wide
   ```

   Example output:

   ```
   NAME       WEIGHT ...  PHASE   ENABLED   DISABLED MESSAGE   READY
   registry   38     ...  Ready   True                         True
   ```

2. Ensure the Deckhouse job queue is empty and contains no errors:

   ```
   d8 system queue list
   ```

   Example output:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```

3. Set the `Unmanaged` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

   Configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.io/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Replace with your license key.
```

4. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

```
conditions:
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

5. If you need to switch back to the old registry management method, refer to the instruction.

⚠ This is a deprecated format for registry management.

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The registry-packages-proxy module

Included in extensions: **Advanced Infrastructure Security**

The module lifecycle stage: General Availability

The `registry-packages-proxy` module provides an in-cluster HTTP proxy service for accessing packages from container registries. It acts as an intermediary between cluster components and external or internal registries, offering caching capabilities to optimize bandwidth usage and improve package retrieval performance.

This module is a critical infrastructure component that runs on master nodes and is used during cluster bootstrap and runtime operations to fetch packages from container registries.

The module deploys a highly-available proxy service that:

- Runs as a deployment on master nodes with `hostNetwork` enabled to ensure availability during bootstrap when CNI is not yet available.
- Listens on port `4219` (HTTPS) on each master node's IP address.
- Provides a `/package` endpoint for retrieving registry packages by digest.
- Implements local caching of retrieved packages (up to 1 GB) to reduce network traffic and improve performance.
- Watches ModuleSource custom resources to obtain registry credentials for different repositories.
- Uses `kube-rbac-proxy` to secure access to the proxy and metrics endpoints.
- Exposes a public HTTPS API (via Ingress) for Deckhouse CLI binaries.
- Exposes an in-cluster HTTPS API for package icons (no public Ingress).

Architecture

The proxy service consists of two containers:

1. **registry-packages-proxy**: The main proxy application that:
   - Fetches packages from remote registries using digests.
   - Caches packages locally in an ephemeral volume (1 GB max).
   - Supports authentication to registries via credentials from ModuleSource resources.
   - Provides health checks and Prometheus metrics.
   - Listens on `127.0.0.1:5080` (HTTP, internal).
2. **kube-rbac-proxy**: Provides RBAC-based access control:
   - Exposes the service on port `4219` (HTTPS).
   - Secures `/metrics` endpoint with Kubernetes RBAC authorization.
   - Secures `/package` endpoint requiring appropriate permissions.
   - Secures `/v1/images/*` (Deckhouse CLI downloads) with Kubernetes RBAC authorization.
   - Allows unauthenticated access to `/healthz`.

HTTP API

After the cluster is bootstrapped and the DNS name template is configured in the publicDomainTemplate parameter, the module creates an Ingress for `registry-packages-proxy` using that template (for example, `registry-packages-proxy.company.my` for `publicDomainTemplate: "%s.company.my"`).

The endpoints listed below are reachable in two ways:

- **Public** (via the Ingress on the `registry-packages-proxy` public host): only `/v1/images/*`.
- **In-cluster only** (via the `registry-packages-proxy.d8-cloud-instance-manager.svc` Service on port `443`, or `:4219` on each master node for bootstrap): all routes, including `/v1/packages/*`.

Package icons (`/v1/packages/`) — in-cluster only

Package icons are served **without authentication** (`kube-rbac-proxy` excludes these paths from RBAC) but are **only reachable from inside the cluster**. They are intentionally **not** routed through the public Ingress, so the public domain `registry-packages-proxy.<publicDomain>` will not serve them.

Method	Path	Description
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon/`	Icon of the latest semver tag
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon`	Same as above
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon/<VERSION>`	Icon of a specific version (`<VERSION>` is a semantic version, e.g. `v1.0.1`)

`<PACKAGE-REPOSITORY>` is the `metadata.name` of a PackageRepository custom resource; its `spec.registry.repo` field tells the proxy which registry path to pull the icon from. The proxy reads the icon from the OCI image `<spec.registry.repo>/<PACKAGE-NAME>:<TAG>`.

The proxy looks for the following files inside the package image, in this priority order, and returns the first one it finds:

Path	Content-Type
`docs/icon.svg`	`image/svg+xml`
`docs/icon.png`	`image/png`
`docs/icon.jpg`	`image/jpeg`
`docs/icon.jpeg`	`image/jpeg`

If none of these are present (or the file is larger than 4 MiB), the proxy returns `404 Not Found`; callers should fall back to a default icon. SVG is preferred because it is resolution-independent.

Example request from a pod in the cluster:

```
curl -fsSk "https://registry-packages-proxy.d8-cloud-instance-
manager.svc/v1/packages/my-repo/my-module/metadata/icon/"
```

Example response headers (when the image ships `docs/icon.svg`):

```
Content-Type: image/svg+xml
Content-Disposition: attachment; filename="<PACKAGE-NAME>.svg"
```

Deckhouse CLI downloads (`/v1/images/`) — public

These endpoints are reachable through the public Ingress (`registry-packages-proxy.<PUBLIC_DOMAIN>`) and require a valid Kubernetes token or a client certificate accepted by `kube-rbac-proxy`. The account must have a RBAC permission to `get` the `deployments/cli-binary` subresource `registry-packages-proxy` in the namespace `d8-cloud-instance-manager`.

To grant the required permission, use the ClusterRole `d8:registry-packages-proxy:cli-download`. For details, refer to “Granting access to Deckhouse CLI downloads”.

The same permission is used to download the Deckhouse CLI plugins. The plugins are available at `deckhouse-cli/plugins/<PLUGIN>` through the same `/v1/images/` endpoints.

Method	Path	Description
`GET`	`/v1/images/<IMAGE>/tags`	Returns a list of tags in JSON format
`GET`, `HEAD`	`/v1/images/<IMAGE>/images/<VERSION>`	Returns an OCI image in the `application/x-gzip` format (with flattened layers)
`GET`	`/v1/images/<IMAGE>/manifests/<REFERENCE>`	Returns an image manifest

Allowed `<IMAGE>` values:

- `deckhouse-cli`
- `deckhouse-cli/plugins/<PLUGIN>`, where `<PLUGIN>` is a single path segment

Deckhouse CLI artifacts can live at one of two places. The official registry publishes them once for all editions, at the registry root one level above the cluster’s edition repository. A registry filled by `d8 mirror push` keeps them right under the cluster repository, exactly as pushed.

The proxy takes the repository from the `d8-system/deckhouse-registry` secret and probes the candidate roots in order: first the repository as is, then the repository without the trailing edition segment (`ce`, `be`, `se`, `se-plus`, `ee`, `fe`). The root that answers is remembered and tried first afterwards; a 404 on the remembered root or a repository change in the secret triggers a re-probe. For example, a cluster with the `registry.deckhouse.io/deckhouse/ee` repository reads the images from `registry.deckhouse.io/deckhouse/deckhouse-cli` and `registry.deckhouse.io/deckhouse/deckhouse-cli/plugins/<PLUGIN>`, while a cluster mirrored to `registry.local/dest/ee` reads them from `registry.local/dest/ee/deckhouse-`

`cli/plugins/<PLUGIN>`. A repository without an edition segment is used as is. The credentials come from the same secret in all cases.

The `/v1/images/<IMAGE>/images/<VERSION>` endpoint accepts an optional `platform=<OS>-<ARCH>` query parameter and picks the matching child manifest from a multi-platform index. If the parameter isn't set, the `linux/amd64` platform is used.

The following is an example of a query for a list of Deckhouse CLI tags:

```
curl -fsS \
  -H "Authorization: Bearer ${TOKEN}" \
  "https://registry-packages-proxy.example.com/v1/images/deckhouse-cli/tags"
```

Internal `/package` endpoint

The legacy `/package?digest=...` endpoint (used during bootstrap and by internal components) remains protected by RBAC (`deployments/http` subresource). It is not exposed through the public Ingress.

Package retrieval flow

When a component requests a package:

1. The request includes a `digest` parameter (required) and optional `repository` and `path` parameters.
2. The proxy checks its local cache for the requested digest.
3. If cached, the package is served directly from cache.
4. If not cached:
   - The proxy fetches credentials for the specified registry from watched ModuleSource resources.
   - The package is retrieved from the remote registry.
   - The package is streamed to the client while simultaneously being cached for future requests.
5. Responses include appropriate HTTP headers for caching (`Cache-Control`, `ETag`, `Content-Length`).

High availability

The module ensures high availability through:

- Running multiple replicas on master nodes (in HA configurations).
- Pod anti-affinity rules to distribute pods across different masters.
- PodDisruptionBudget to prevent simultaneous disruption of all replicas.
- Vertical Pod Autoscaler support for automatic resource adjustments.

RBAC roles provided by the module

ClusterRole	Purpose
`d8:registry-packages-proxy:cli-download`	Access to `/v1/images/*`: the `deckhouse-cli` binary and its plugins
`d8:registry-packages-proxy:packages-download`	Reserved for future authenticated `/v1/packages/*` routes (icons are served anonymously, in-cluster only)

Neither role is bound to anyone out of the box.

Granting access to Deckhouse CLI downloads

For Deckhouse CLI to work, a user requires permissions to download binaries and obtain the proxy public address.

> ⚠ To be able to create ClusterRoleBinding or RoleBinding, the account must have permissions to assign a respective role. Use the account with the corresponding permissions if necessary, such as `/etc/kubernetes/super-admin.conf`.

1. Grant access to download binaries. To do that, use the ClusterRole `d8:registry-packages-proxy:cli-download`. This role grants permissions required to self-update Deckhouse CLI (`d8`

`cli`) and download plugins (`d8 plugins`):

Create a ClusterRoleBinding:

```
d8 k create clusterrolebinding d8-cli-download \
  --clusterrole=d8:registry-packages-proxy:cli-download \
  --group=<GROUP>
```

Instead of `--group=<GROUP>`, you can grant permissions to a specific user (`--user=<EMAIL>`) or service account (`--serviceaccount=<NAMESPACE>:<NAME>`).

2. Grant access to obtain public address of the proxy. Deckhouse CLI obtains public address from the `registry-packages-proxy` Ingress.
   - Create the `d8-cli-ingress` Role, which grants the `get` permission for that Ingress:

     ```
     d8 k -n d8-cloud-instance-manager create role d8-cli-ingress \
       --verb=get \
       --resource=ingresses \
       --resource-name=registry-packages-proxy
     ```

   - Create a RoleBinding for assigning the created role to a target group:

     ```
     d8 k -n d8-cloud-instance-manager create rolebinding d8-cli-ingress \
       --role=d8-cli-ingress \
       --group=<GROUP>
     ```

   Without granted access, the proxy address needs to be specified manually:

   ```
   d8 cli check --rpp-endpoint https://registry-packages-proxy.<PUBLIC_DOMAIN>
   ```

3. Check the granted permissions without logging in as the user. The permissions are limited by `resourceNames`, therefore make sure to specify the resource name.
   - Ensure the access to download binaries has been granted:

     ```
     d8 k auth can-i get deployments/registry-packages-proxy \
       -n d8-cloud-instance-manager \
       --subresource=cli-binary \
       --as=<EMAIL_OR_SERVICE_ACCOUNT>
     ```

   - Ensure the access to obtain Ingress `registry-packages-proxy` has been granted:

     ```
     d8 k auth can-i get ingresses/registry-packages-proxy \
       -n d8-cloud-instance-manager \
       --as=<EMAIL_OR_SERVICE_ACCOUNT>
     ```

   `Kube-rbac-proxy` caches authorization decisions: a denial for 30 seconds, an approval for 5 minutes. Consider that while checking the granted permissions: once the permission has been granted, a `403` response clears in half a minute, and after a permission has been revoked, the access is still granted for up to 5 minutes.

Limitations

- The module runs exclusively on master nodes.
- It requires `hostNetwork: true` to function during bootstrap phase.
- Cache size is limited to 1 GB per pod.
- Most HTTP endpoints require Kubernetes RBAC; only health checks (healthz) and package icons are anonymous. Package icons are additionally limited to in-cluster access (no public Ingress route).
- Package icons are read from fixed paths inside the package image (`docs/icon.svg`, `docs/icon.png`, `docs/icon.jpg`, `docs/icon.jpeg`) with SVG preferred. Maximum icon size is 4 MiB.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The registry-packages-proxy: configuration

> Included in extensions: **Advanced Infrastructure Security**

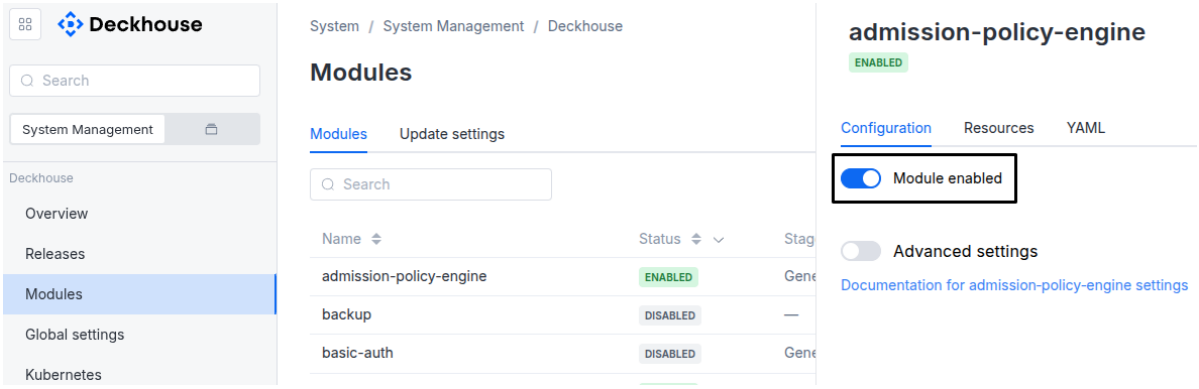
> The module lifecycle stage: General Availability

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable registry-packages-proxy
```

- **Using ModuleConfig `registry-packages-proxy`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `registry-packages-proxy` (create it if necessary);

  Example of a manifest to enable module `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  enabled: true
```

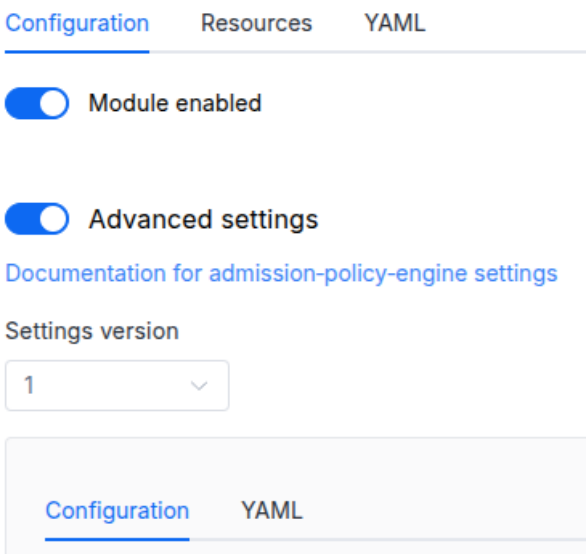
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `registry-packages-proxy` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `registry-packages-proxy`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `registry-packages-proxy` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc registry-packages-proxy
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `registry-packages-proxy` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `registry-packages-proxy`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: registry-packages-proxy
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

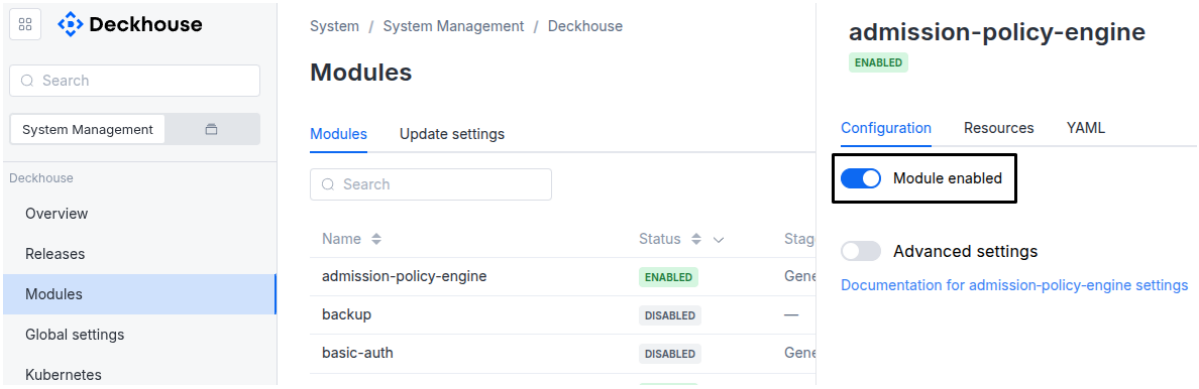
The module does not have any mandatory parameters.

The module is **enabled** by default in the `Default`, `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`, `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable registry-packages-proxy
```

- **Using ModuleConfig `registry-packages-proxy`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `registry-packages-proxy` (create it if necessary);

  Example of a manifest to enable module `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  enabled: true
```

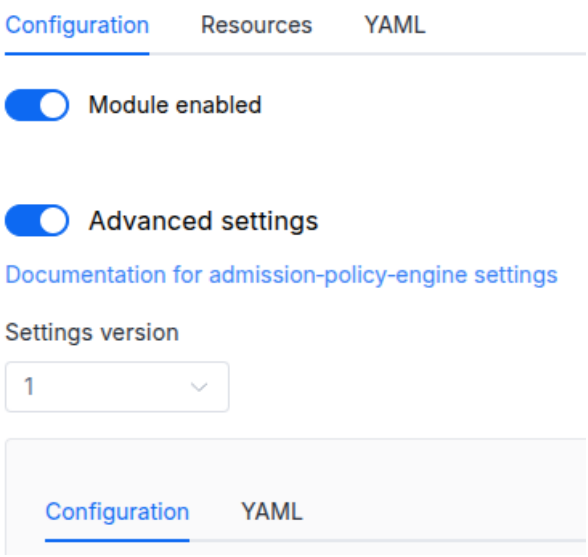
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `registry-packages-proxy` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `registry-packages-proxy`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `registry-packages-proxy` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc registry-packages-proxy
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `registry-packages-proxy` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `registry-packages-proxy`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: registry-packages-proxy
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Example:

```
highAvailability: true
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The secret-copier module

> The module lifecycle stage: General Availability

This module copies Secrets to all namespaces.

It facilitates the copying of Secrets required to pull images and to provision Ceph's RBDs as part of the CI process.

> ⚠ The `secret-copier` module cannot be used together with `multitenancy-manager`.
>
> `multitenancy-manager` creates isolated environments for users within their projects, while `secret-copier` automatically distributes secrets across all namespaces. This may lead to sensitive data leaks if important secrets end up in a user's isolated environment, violating the security model.
>
> If you need to provide a shared certificate (e.g., a WC certificate for an internal environment) or a shared registry access token, do not use `secret-copier`. Instead, place such secrets in the project template in `multitenancy-manager` — the cluster administrator should define them in the project configuration.

How does it work?

This module monitors the Secrets (with the `secret-copier.deckhouse.io/enabled: ""` label) in the `default` namespace for changes.

- The module copies such a Secret to all namespaces after it is created.
- When a Secret is changed, its new contents are also propagated to all namespaces.
- When a Secret is deleted, its copies are deleted from all namespaces.
- Copies have the `heritage: deckhouse` label, so users, including the cluster administrator, cannot modify or delete them. To change or delete a copy, change or delete the source Secret in the `default` namespace.
- If a copy still differs from the source Secret, the module overwrites it with the original content.
- The module copies all the Secrets (that have the `secret-copier.deckhouse.io/enabled: ""` label) of the `default` namespace to any newly created namespace.
- The module never creates, updates or deletes the Secret `d8-system/cluster-key`, which holds the cluster key of Deckhouse Platform licensing.
- The module does not copy Secrets of the `bootstrap.kubernetes.io/token` type. The copies of such Secrets that earlier versions of the module made are deleted.
- The module updates and deletes only its own copies. If a namespace already has a Secret with the same name that the module did not create, the module leaves it as is and does not copy the Secret into that namespace.

Additionally, it synchronizes the Secrets every night, ensuring they are identical to those in the `default` namespace.

What do I need to configure?

All you need to do is to create a Secret with the `secret-copier.deckhouse.io/enabled: ""` label in the `default` namespace.

> **Note!** The working namespace for the module is `default`, Secrets will be copied only from it. Secrets with the secret-copier.deckhouse.io/enabled: “” label created in other namespaces will be automatically deleted when the module is enabled.

How to synchronize Secret to some selected namespaces instead of all namespaces?

Specify namespace label-selector in the value of the `secret-copier.deckhouse.io/target-namespace-selector` annotation. For example: `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. The module will create a copy of that Secret in all namespaces that matches the label-selector.

If the annotation value is not a valid label selector, the module does not copy the Secret to any namespace and deletes its copies made before.

How do I know that a Secret was not copied?

The `D8SecretCopierSecretNotCopied` alert fires with one of the reasons:

- `InvalidNamespaceSelector`: the `secret-copier.deckhouse.io/target-namespace-selector` annotation of the Secret is not a valid label selector. Fix the annotation value.
- `ForeignSecretExists`: the target namespace already has a Secret with the same name that the module did not create. Delete or rename that Secret.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The terraform-manager module

> The module lifecycle stage: General Availability

The module is responsible for monitoring and synchronizing the state of the underlying infrastructure and persistent nodes in the cloud environment. The implementation is based on Terraform and is used in the Deckhouse Platform for interaction with supported cloud providers.

- The module consists of 2 parts:
  - `terraform-auto-converger` — checks the Terraform state and applies non-destructive changes;
  - `terraform-state-exporter` — checks the Terraform state and exports cluster metrics.
- The module is enabled by default if the following secrets are present in the cluster:
  - `kube-system/d8-provider-cluster-configuration`;
  - `d8-system/d8-cluster-terraform-state`.

Third-party components

List of third-party software used in the `terraform-manager` module:

- Terraform

  License: Mozilla Public License 2.0

  A tool for building, changing, and versioning infrastructure safely and efficiently.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

> The module lifecycle stage: General Availability

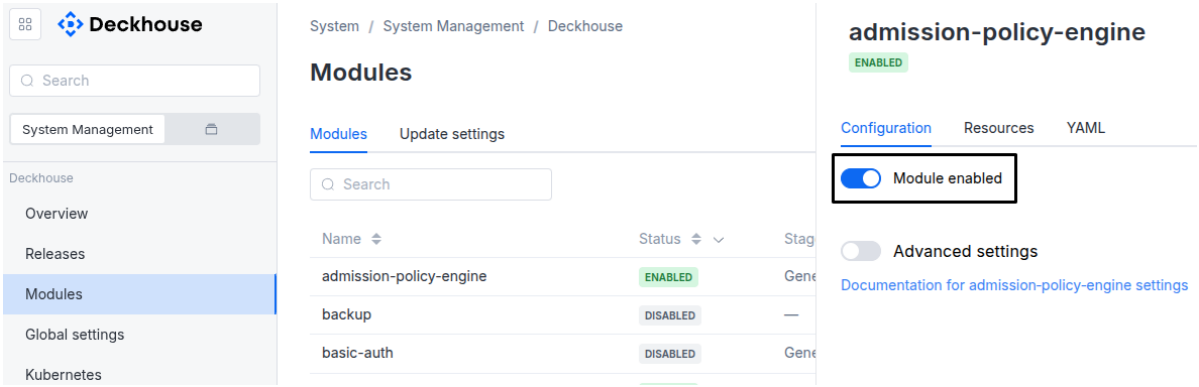
The module has 10 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `terraform-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable terraform-manager
```

- **Using ModuleConfig `terraform-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `terraform-manager` (create it if necessary);

  Example of a manifest to enable module `terraform-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: terraform-manager
spec:
  enabled: true
```

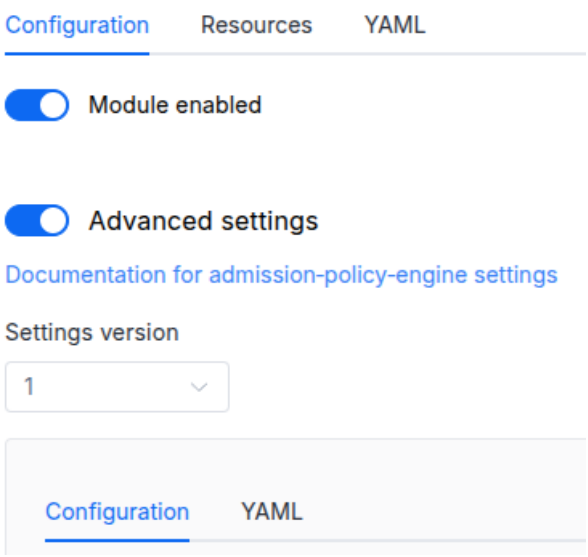
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `terraform-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `terraform-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `terraform-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `terraform-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc terraform-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `terraform-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `terraform-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: terraform-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.autoConvergerEnabled**
    boolean

    Enables periodic checking of Terraform state and applying non-destructive changes.

    The frequency of the check is set by the autoConvergerPeriod parameter.

    Default: `true`

- **settings.autoConvergerPeriod**
  string

  The time interval after which the state of Terraform is checked and applied.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  Default: `1h`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.nodeSelector**
  object

  The same as in the Pod's `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Module user-authn

> Included in extensions: **Billing**

> The module lifecycle stage: General Availability

The module sets up a unified authentication system integrated with Kubernetes and Web interfaces used in other modules (Grafana, Dashboard, etc.).

Main features

The module provides the following features:

- Single sign-on to the web interfaces of Deckhouse Platform (DP) modules and to the Kubernetes API through Dex
- Sign-in through external providers: GitHub, GitLab, Bitbucket Cloud, Atlassian Crowd, LDAP (with Kerberos SSO), OIDC and SAML 2.0
- Static users and groups with a password policy, account lockout and two-factor authentication
- Authentication for any web application in the cluster with DexAuthenticator, and OAuth2 clients for applications that support OIDC (DexClient)
- Kubeconfig generation in the Deckhouse web UI for access with `kubectl` and OIDC tokens
- Basic authentication to the Kubernetes API through one Crowd, OIDC or LDAP provider

Components

The module consists of the following components:

- dex — is a federated OpenID Connect provider that acts as an identity service for static users and can be linked to one or more ID providers (e.g., SAML providers, GitHub, and GitLab);
- `dex-authenticator` (based on oauth2-proxy) authenticates requests to an application published through an Ingress or, with the `alb` module, through Gateway API. It runs in the `d8-user-authn` namespace.

Management of static users is performed using the User and Group resources:

- The User object stores user information, including the email address and a password hash (the plain-text password is not stored);
- The Group object defines a list of users grouped together for easier access management.

User management operations — creating and deleting users, managing groups, resetting passwords, resetting 2FA, and locking/unlocking — can be performed via the CLI using the `d8 iam` command group or via the UserOperation resource. The `UserOperation` resource is single-use and is automatically deleted after `retention.userOperations` (24 hours by default).

The following external authentication protocols/providers are supported:

- GitHub;
- GitLab;
- BitBucket Cloud;
- Crowd;
- LDAP;
- OIDC;
- SAML 2.0.

You can use several external authentication providers simultaneously.

Integration features

OIDC authentication in `kube-apiserver` is configured automatically by the `control-plane-manager` module. For details, see Configuring kube-apiserver.

⚠ When the module is enabled, the web interfaces of DP modules authenticate users through Dex instead of HTTP Basic Auth.

Basic authentication for the Kubernetes API requests

Basic authentication for Kubernetes API requests is available for the Crowd, OIDC and LDAP providers (the `enableBasicAuth` parameter in the crowd, oidc or ldap section). It can be enabled for only one provider in the cluster. For a Crowd or OIDC provider, Basic Authentication bypasses the identity limits of the provider, so only a SuperAdmin can enable it.

i You can also connect to the Kubernetes API via other supported external providers.

Application integration

To add authentication to any web application in the cluster, create a DexAuthenticator in the namespace of the application and add the authentication annotations to its Ingress or HTTPRoute. DexAuthenticator can limit the groups and emails that are allowed to sign in and the networks from which users can authenticate.

The `Served` condition in the status of the object shows whether the module serves it (see Checking that a DexAuthenticator is served).

If the application supports OIDC itself, create a DexClient in its namespace instead. The module registers an OAuth2 client in Dex and creates the `dex-client-<NAME>` Secret with the client secret in the same namespace. DexClient can limit the groups and emails that are allowed to sign in and list the clients whose tokens the application trusts (`trustedPeers`).

Generating kubeconfig files

After signing in, a user can generate a kubeconfig for kubectl and other Kubernetes tools in the Deckhouse web UI. The kubeconfig uses an OIDC token. If the authentication provider supports session extension, the kubeconfig also contains a refresh token, and the session is extended automatically. If

the provider reports during the extension that the user no longer exists or has lost access, Dex refuses the refresh token with the `invalid_grant` error, and the user has to sign in again.

The `kubeconfigGenerator` parameter adds other ways to reach the API server, each with its own address and CA certificate, for example, for access through a bastion host or a VPN.

Additional Dex capabilities

The module ships Dex with additional capabilities:

- Groups for static users (the Group object)
- Two-factor authentication (2FA) for local users and the users of LDAP and Crowd providers (the `twoFactor` parameter)
- A password policy for static users: complexity, password history, rotation and lockout after failed sign-in attempts (the `passwordPolicy` parameter)
- A forced password change at the next sign-in after an administrator resets the password
- Lockout after failed sign-in attempts for LDAP and Atlassian Crowd users (the `passwordPolicy.lockout.applyToConnectors` parameter)
- Per-IP rate limiting of failed sign-in attempts (the `rateLimit` parameter)
- Locking and unlocking users of any provider with the UserOperation object, also without a password policy
- Lists of allowed users and groups in DexClient and DexAuthenticator that Dex checks at sign-in and at every refresh-token request
- Limits on the identities a provider may assert (the `spec.allowedIdentities` parameter of DexProvider)
- Kerberos (SPNEGO) sign-in through the LDAP provider without the password form, with tickets from the `Authorization: Negotiate` header validated by the service keytab
- Refresh tokens for the SAML provider
- A custom CA certificate for the GitLab provider (the `gitlab.rootCAData` parameter)

High availability mode

In high availability mode (`highAvailability`), Dex and the other module components run in several replicas, so sign-in keeps working if one pod fails. dex-authenticator always runs in two replicas.

Third-party components

List of third-party software used in the `user-authn` module:

- Dex 2.46.0

  License: Apache License 2.0

  A Federated OpenID Connect Provider with pluggable connectors.

- OAuth2 Proxy 7.15.5

  License: MIT License

  A reverse proxy and static file server that provides authentication using Providers (Google, GitHub, and others) to validate accounts by email, domain or group.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The user-authn module: configuration

> Included in extensions: **Billing**

> The module lifecycle stage: General Availability

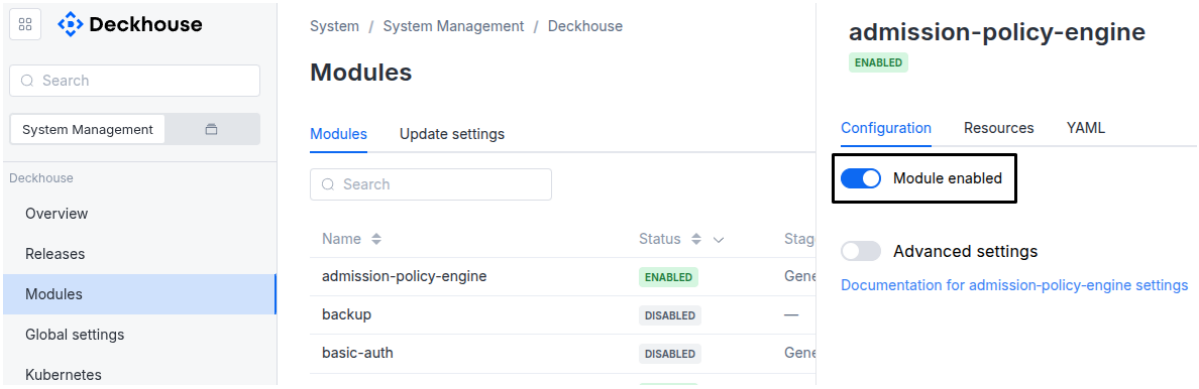
The module has 25 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authn` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable user-authn
```

- **Using ModuleConfig `user-authn`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `user-authn` (create it if necessary);

  Example of a manifest to enable module `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  enabled: true
```

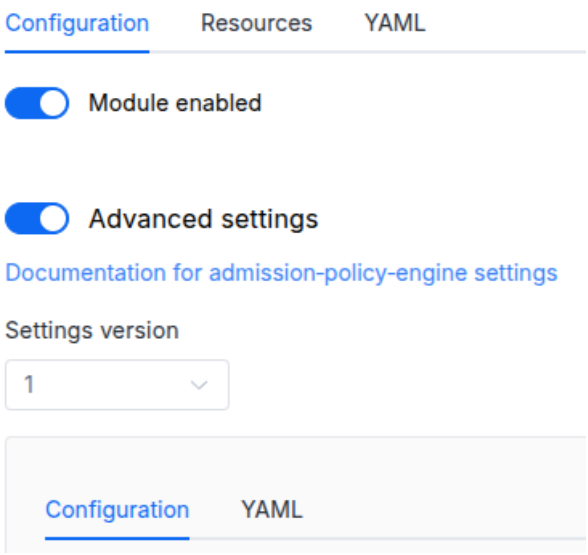
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authn` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `user-authn` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `user-authn`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `user-authn` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc user-authn
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `user-authn` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `user-authn`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: user-authn
  spec:
    version: 4
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Replace `publishAPI.enable` with `publishAPI.enabled`.

- Updates from version **2** to **3**:

  An `idTokenTTL` of 6 hours or more is replaced with `5h59m`: Dex rotates its signing keys every 6 hours.

- Updates from version **3** to **4**:

  The `staticUsers2FA` parameter is renamed `twoFactor`. The second factor can now be turned on for more than the local users.

Parameters

Schema version: 4

- **settings**
  object
  - **settings.controlPlaneConfigurator**
    object

    Parameters of the `control-plane-manager` module for automatic configuration of `kube-apiserver`.

    - **settings.controlPlaneConfigurator.dexCAMode**
      string

      How to determine the CA that will be used when configuring `kube-apiserver`.

      - `Custom` — use the CA explicitly set via the `dexCustomCA` parameter (see below). This option comes in handy if you use an external HTTPS load balancer in front of Ingresses, and this load balancer relies on a self-signed certificate.
      - `DoNotNeed` — a CA is not required (e.g., when using a public LE or other TLS providers).
      - `FromIngressSecret` — extract the CA of certificate from the Secret that is used in the Ingress. This option comes in handy if you use self-signed certificates with Ingresses.

      Default: `DoNotNeed`

      Allowed values: `Custom`, `DoNotNeed`, `FromIngressSecret`

    - **settings.controlPlaneConfigurator.dexCustomCA**
      string

      The CA to use if `dexCAMode` is `Custom`. Plain text (no base64).

    - **settings.controlPlaneConfigurator.enabled**
      boolean

      Defines if the `control-plane-manager` module should be used to configure OIDC for the `kube-apiserver`.

      Default: `true`

  - **settings.devices**
    object

    Devices of a user: the browsers and the other clients (kubeconfig, `d8 login`) the user signed in from.

    - **settings.devices.maxPerApplication**
      integer

      The maximum number of devices on which a user can be simultaneously logged in to a single application (DexAuthenticator, DexClient, kubeconfig).

      A device is a browser with a Dex session together with all application domains it has logged in to, or a refresh token without a browser session, e.g. a kubeconfig file.

      If a user logs in from a new device and the limit specified in the parameter has been reached, the device that has been unused the longest is logged out of the application. Non-trusted devices are logged out first (for details, see the description of the `trustedAfter` parameter).

      Default: `10`

      Allowed values: `1 <= X <= 1000`

      Examples:

      ```
      maxPerApplication: 10
      ```

```
maxPerApplication: 3
```

- **settings.devices.rememberedBrowsers**
  integer

  How many browsers of a user Dex remembers with their first sign-in.

  A remembered browser that signs in again keeps its age: the time until it is trusted (`trustedAfter`) counts from its first sign-in. Past the limit, the browser seen longest ago is forgotten and counts as new at its next sign-in.

  `0` remembers no browser: every sign-in is a new device.

  Default: `20`

  Allowed values: `0 <= X <= 1000`

  Examples:

  ```
  rememberedBrowsers: 20
  ```

  ```
  rememberedBrowsers: 0
  ```

- **settings.devices.trustedAfter**
  string

  Time after which a device of the user is trusted, counted from the first sign-in from its browser (from the sign-in for a device without a browser).

  Only a trusted device lets the user sign out of all devices, of the other devices or of another device, and request a password change on their own. A device signed in more recently can sign the user out only of itself. Someone who has just signed in with a stolen password therefore can neither sign the owner out nor change the password, while the owner sees the new device on a trusted one and signs it out. A stolen session is not covered: a copied cookie of Dex or of the web interface has the age of the owner’s device. The operations of an administrator are not limited.

  When an application reaches the limit of devices of a user, the devices that are not trusted yet are signed out of it first.

  `0s` trusts every device at once: any device can sign out the others and request a password change.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

  Default: `24h`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

  Examples:

  ```
  trustedAfter: 24h
  ```

  ```
  trustedAfter: 1h
  ```

  ```
  trustedAfter: 0s
  ```

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Learn more about the high availability mode for modules.

  Examples:

  ```
  highAvailability: true
  ```

```
highAvailability: false
```

- **settings.https**
  object

  What certificate type to use with Dex.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object

    Settings of the `CertManager` mode.

    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for Dex.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**
    object

    Settings of the `CustomCertificate` mode.

    - **settings.https.customCertificate.secretName**
      string

      The name of the Secret in the `d8-system` namespace to use with Dex.

      This Secret must have the kubernetes.io/tls format.

      Default: `false`

  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `CertManager`: Dex uses HTTPS and gets a certificate from the ClusterIssuer set in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate`: Dex uses HTTPS with the certificate from the `d8-system` namespace.
    - `Disabled`: The module is turned off, because it works only with HTTPS.
    - `OnlyInURI`: Dex works over HTTP behind an external load balancer that terminates HTTPS. All `user-authn` links use the HTTPS scheme. The load balancer must redirect HTTP to HTTPS.

    If you set the `https` section without `mode`, `Disabled` applies and the module is turned off.

    Default: `Disabled`

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.idTokenTTL**
  string

  TTL of the ID token

TTL of the ID token.

Set as a string specifying hours, minutes, or seconds: `30m`, `20s`, `2h30m10s`, `5h`.

Must be greater than zero and strictly less than 6 hours (Dex signing keys rotate every 6 hours). A ModuleConfig of an earlier settings version (`spec.version` 1 or 2) with a value of 6 hours or more is still accepted: its value is converted to `5h59m`.

Defines the minimum possible user session lifetime (via `keepUsersLoggedInFor` in the DexAuthenticator resource and parameters like `auth.sessionTTL`, for example, in the `console` module).

If the session TTL is less than or equal to `idTokenTTL`, the effective logout time is `idTokenTTL + 1s`.

To force logout sooner than the default `idTokenTTL`, decrease this parameter value as well.

Default: `10m`

Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.ingressClass**
  string

  The class of the Ingress controller that will be used for Dex.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.kubeconfigGenerator**
  array of objects

  An array in which additional possible methods for accessing the API server are specified.

  This option comes in handy if you prefer not to grant access to the cluster's API via Ingress but rather do it by other means (e.g., using a bastion host or over OpenVPN).

  - **settings.kubeconfigGenerator.description**
    string

    A couple of words how this authentication method differs from others.

  - **settings.kubeconfigGenerator.id**
    string

    Required value

    The name of the method for accessing the API server (no spaces, lowercase letters).

    Pattern: `^[\@\.\:0-9a-z._-]+$`

  - **settings.kubeconfigGenerator.masterCA**
    string

    A CA for accessing the API:

    - If the parameter is not set, Kubernetes CA is used.
    - We recommend using a self-signed certificate (and specify it as masterCA) if an HTTP proxy (that terminates HTTPS traffic) is used for exposing.

  - **settings.kubeconfigGenerator.masterURI**
    string

    Required value

    API server address that kubeconfig files generated for this access method point to, for example `https://192.168.0.15:6443`.

    If clients reach the API server through a TCP proxy or a load balancer, add its address to the `apiserver.certSANs` parameter of the `control-plane-manager` module.

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted, it is determined automatically.

- **settings.passwordPolicy**

object

Password policy for local users.

Default:

- **settings.passwordPolicy.complexityLevel**
  string

  Password complexity level.

  Depending on the complexity level, a password must meet the following minimum requirements:

  - `None`: 1 character.
  - `Low`: 8 characters.
  - `Fair`: 8 characters, 1 uppercase letter (`A-Z`), 1 lowercase letter (`a-z`), 1 digit.
  - `Good`: 8 characters, 1 uppercase letter (`A-Z`), 1 lowercase letter (`a-z`), 1 digit, 1 special character (`!@#$%^&*`).
  - `Excellent`: 8 characters, 1 uppercase letter (`A-Z`), 1 lowercase letter (`a-z`), 1 digit, 1 special character (`!@#$%^&*`), no more than 2 identical characters in a row.
  - `Custom`: requirements are defined by the `custom` field.

  Default: `Fair`

  Allowed values: `None`, `Low`, `Fair`, `Good`, `Excellent`, `Custom`

- **settings.passwordPolicy.custom**
  object

  Custom password complexity rules.

  Applied only when `complexityLevel` is set to `Custom`. Any combination of the rules below can be enabled independently.

  - **settings.passwordPolicy.custom.capitalized**
    boolean

    If `true`, a password must contain at least one uppercase letter (`A-Z`).

    Default: `false`

  - **settings.passwordPolicy.custom.minLength**
    integer

    Minimum number of characters in a password.

    Default: `8`

    Allowed values: `1 <= X`

  - **settings.passwordPolicy.custom.numbers**
    boolean

    If `true`, a password must contain at least one digit.

    Default: `false`

  - **settings.passwordPolicy.custom.repeatedChars**
    boolean

    If `true`, a password must not contain more than 2 identical characters in a row (the same rule the `Excellent` level applies).

    Default: `false`

  - **settings.passwordPolicy.custom.specialCharacters**
    boolean

    If `true`, a password must contain at least one special character (anything that is neither a letter nor a digit).

    Default: `false`

- **settings.passwordPolicy.lockout**
  object

  Automatic temporary lockout after consecutive failed login attempts.

This section only controls lockout after a failed password. An administrator lock via UserOperation (`d8 iam user lock`) works independently and does not require `passwordPolicy` or this `lockout` block.

- **settings.passwordPolicy.lockout.applyToConnectors**
  array of strings

  External (non-local) connector types the lockout additionally applies to.

  The built-in `Local` connector is always covered by the lockout policy when this section is configured and does not need to be listed here.

  For the listed connector types, Dex counts failed attempts and locks each pair of a provider and a user separately. The UserAccount object of the user shows the counter and the lock.

  Manually unlocking a user is supported for every provider type via a UserOperation.

  - **Element of the array**
    string

    Allowed values: `LDAP`, `Crowd`

- **settings.passwordPolicy.lockout.lockDuration**
  string

  Required value

  Temporary lockout duration. Can be set in seconds, minutes, hours, or days: `10s`, `5m`, `1h`, `3d`.

  Pattern: `^(?:(?:[1-9]|[1-9][0-9]+)d)?(?:(?:[1-9]|[1-9][0-9]+)h)?(?:(?:[1-9]|[1-9][0-9]+)m)?(?:(?:[1-9]|[1-9][0-9]+)s)?$`

- **settings.passwordPolicy.lockout.maxAttempts**
  integer

  Required value

  Number of consecutive failed login attempts after which the user will be temporarily locked out.

- **settings.passwordPolicy.passwordHistoryLimit**
  integer

  Number of the user’s previous passwords stored in history to prevent their reuse.

  Default: `5`

  Allowed values: `0 <= X`

- **settings.passwordPolicy.rotation**
  object

  Password rotation settings.

  - **settings.passwordPolicy.rotation.interval**
    string

    Required value

    Time interval after which the password must be changed. Can be set in seconds, minutes, hours or days: `10s`, `5m`, `1h`, `3d`.

    Pattern: `^(?:(?:[1-9]|[1-9][0-9]+)d)?(?:(?:[1-9]|[1-9][0-9]+)h)?(?:(?:[1-9]|[1-9][0-9]+)m)?(?:(?:[1-9]|[1-9][0-9]+)s)?$`

- **settings.publishAPI**
  Deprecated
  object

  Obsolete. The module does not publish the Kubernetes API. The API is published with the `apiserver.publishAPI` parameter of the `control-plane-manager` module. Move existing settings there.

  For more information on how to expose the Kubernetes API, see the `control-plane-manager` module documentation.

- **settings.publishAPI.addKubeconfigGeneratorEntry**
  boolean

  Obsolete. Use the `apiserver.publishAPI.ingress.addKubeconfigGeneratorEntry` parameter of the `control-plane-manager` module.

  Default: `true`

- **settings.publishAPI.enabled**
  boolean

  Obsolete. Does not create an Ingress.

  Default: `false`

- **settings.publishAPI.https**
  object

  The HTTPS mode for the API server Ingress.

  Examples:

  ```
  https:
    mode: SelfSigned
  ```

  ```
  https:
    mode: Global
    global:
      kubeconfigGeneratorMasterCA: plainstring
  ```

  - **settings.publishAPI.https.global**
    object

    An additional parameter for the `Global` mode.

    - **settings.publishAPI.https.global.kubeconfigGeneratorMasterCA**
      string

      If there is an external load balancer in front of the Ingress that terminates HTTPS traffic using non-public CA, then you need to specify the CA so it will be included in kubectl-config.

      If you are using certificates issued by the `cert-manager` module and Let's Encrypt in your cluster, you should set an empty string `""` as the value.

      Also, you can set the external LB's certificate itself as a CA if you can't get the CA that signed it for some reason. Note that after the certificate is updated on the LB, all the previously generated kubeconfigs will stop working.

  - **settings.publishAPI.https.mode**
    string

    The mode of issuing certificates for the Ingress resource.

    In the `SelfSigned` mode, a CA-signed certificate will be issued for the Ingress resource.

    In the `Global` mode, the policies specified in the `global.modules.https.mode` global parameter will be applied. Thus, if the global parameter has the `CertManager` mode set (with `letsencrypt` as the ClusterIssuer), then the Let's Encrypt certificate will be issued for the Ingress resource.

    Default: `SelfSigned`

    Allowed values: `SelfSigned`, `Global`

- **settings.publishAPI.ingressClass**
  string

  The Ingress class that will be used to expose the Kubernetes API via Ingress.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.publishAPI.whitelistSourceRanges**

array of strings

An array of CIDRs that are allowed to connect to the API server.

- **Element of the array**
  string

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))?$`

- **settings.rateLimit**
  object

  Brute-force protection for the password endpoints of Dex.

  When enabled, Dex limits the failed requests from each client IP address to the following endpoints:

  - `/token`: Token endpoint: client authentication and the password grant. Other requests of an authenticated client, such as refresh-token requests, do not count.
  - `/auth/{connector}/login`: Sign-in form of the local, LDAP and Crowd providers.
  - `/password/change`: Password change page.
  - `/mfa` and the paths under it: Steps of two-factor authentication (2FA).

  The client IP address is taken from the `X-Real-IP` header that the platform Ingress controller sets. Dex trusts the header only from the proxies of the platform. The Ingress controller proves itself with its client certificate, which the platform CA issued for the `ingress-nginx:auth` organization. The controller presents it to any backend an Ingress asks it to, so whoever can create an Ingress with configuration snippets on that controller can set the address too. The Gateway sends a secret stored in the HTTPRoute object of the module in `d8-user-authn`: whoever can read it can set the address too. For any other request, for example from a pod that calls the Dex Service directly, Dex counts the address of the connection. Through the Gateway API (the `alb` module) Dex counts the last address of `X-Forwarded-For`: the address the ALB saw, or the one a balancer passed when the ALB instance trusts it in `originalIPDetection.setRealIPFrom`. Behind a balancer the ALB instance does not trust, every client is counted by the address of the balancer. Basic Authentication to the Kubernetes API through an LDAP provider is counted per address of the client that the platform Ingress controller or Gateway saw: basic-auth-proxy passes on the last hop of `X-Forwarded-For`. The Ingress controller has to see the address of the client itself: behind a balancer that replaces it, all users are counted together (see Which client address does Dex see?).

  The limit applies to **failed authentication attempts only**. Each request takes a token from the bucket of its IP address, and a successful response (HTTP `2xx` or `3xx`) returns the token. Successful logins and refresh-token exchanges do not consume the per-IP budget. When the bucket is empty, Dex answers with the `429` status code and the `Retry-After` header.

  Limits are tracked in memory in each Dex replica. In HA setups, the effective per-IP rate is multiplied by the number of replicas. Account lockout complements this limit.

  - **settings.rateLimit.enabled**
    boolean

    Enable per-IP rate limiting on Dex password endpoints.

    Default: `false`

  - **settings.rateLimit.perIP**
    object

    Per-source-IP token-bucket configuration.

    Both `requestsPerMinute` and `burst` parameters count failed authentication attempts only.

    - **settings.rateLimit.perIP.burst**
      integer

      Maximum bucket size defining the allowed number of failed authentication attempts from a single IP address before the throttling activates.

      Default: `10`

      Allowed values: `1 <= X`

    - **settings.rateLimit.perIP.requestsPerMinute**

integer

Maximum rate of failed authentication attempts tolerated per IP address (in attempts per minute).

Successful logins and refresh-token exchanges do not consume the budget.

Default: `10`

Allowed values: `1 <= X`

- **settings.refreshTokenAbsoluteLifetime**
  string

  Maximum lifetime of refresh tokens (maps to Dex `expiry.refreshTokens.absoluteLifetime`).

  This setting controls how long user sessions can persist. After this period expires, users will need to re-authenticate, even if they have active refresh tokens.

  A refresh token that is not used for `refreshTokenValidIfNotUsedFor` expires earlier.

  The value must not be shorter than `idTokenTTL`.

  It is specified as a string containing the time unit in hours, minutes and seconds: 30m, 20s, 2h30m10s, 24h.

  Default: `4380h`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.refreshTokenValidIfNotUsedFor**
  string

  Time after which a refresh token that is not used expires (maps to Dex `expiry.refreshTokens.validIfNotUsedFor`).

  A kubeconfig file or an application that has not refreshed its tokens for this time signs in again. By default, about 3 months.

  The value must be greater than zero and not longer than `refreshTokenAbsoluteLifetime`.

  It is specified as a string containing the time unit in hours, minutes and seconds: 30m, 20s, 2h30m10s, 24h.

  Default: `2190h`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

  Examples:

  ```
  refreshTokenValidIfNotUsedFor: 2190h
  ```

  ```
  refreshTokenValidIfNotUsedFor: 720h
  ```

- **settings.retention**
  object

  How long the records of the module are kept.

  - **settings.retention.revocations**
    string

    How long a record of a revocation (the end of the sessions and tokens of a user by an operation or a sign-out) is kept for the components that end the sessions of the user from it: DexAuthenticator objects, Basic Authentication to the Kubernetes API, the authorization webhook.

    By default, and at least, `idTokenTTL` and an hour, but not less than 70 minutes: until every token issued before the revocation expired. A shorter value is raised to this minimum.

    Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

    Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

    Example:

```
revocations: 24h
```

- **settings.retention.userOperations**
  string

  How long a UserOperation object is kept after its creation. Then it is deleted, and the history of operations of the web interface no longer shows it.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

  Default: `24h`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

  Examples:

  ```
  userOperations: 24h
  ```

  ```
  userOperations: 720h
  ```

- **settings.sessions**
  object

  Browser sessions of Dex.

  With sessions on, Dex remembers the user signed in to a browser:

  - The web interfaces of Deckhouse (the DexAuthenticator objects of the platform modules, such as the console or Grafana) share the sign-in: the user enters the password and the second factor once and opens the others without them (see `sso`). For a user of an external provider, this lasts while the stored user data is fresh (see `identityRefresh`).
  - The sign-in form shows the “Remember me” checkbox. With it, the session survives closing the browser.
  - Signing out of an application of a DexAuthenticator object ends the session: the browser is signed out of all the applications of DexAuthenticator objects it signed in to through this session within seconds.

  What does not change:

  - Access tokens for `kubectl` (kubeconfig, `d8 login`) and applications of the DexClient objects that request offline access are issued only after an interactive sign-in, never from a session of the browser. A DexClient with `refreshTokenLifetime: Session` is the exception.
  - An application signed in through a DexAuthenticator stays signed in for `keepUsersLoggedInFor`, even after the session of Dex expires. The session limits only signing in without a password.
  - Signing out of Dex does not sign the user out of an external OIDC provider unless the provider has `upstreamLogout` enabled.
  - **settings.sessions.absoluteLifetime**
    string

    Maximum lifetime of a session, counted from the sign-in.

    However much the browser is used, the session ends after this time, and the next application asks for the password and the second factor again. By default, a user who works every day enters them once a week.

    Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

    Default: `168h`

    Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

  - **settings.sessions.enabled**
    boolean

    Turns the browser sessions of Dex on.

    Default: `false`

  - **settings.sessions.identityRefresh**
    object

    Freshness of the user data that Dex stores at the sign-in, such as the groups of the user.

Dex serves the stored data until it is older than `maxAge`, and then reads it from the provider again. This holds with sessions on or off: a refresh of the tokens of an application reaches the provider only when the stored data is older than `maxAge`. A user disabled at the provider therefore keeps refreshing the tokens until then. If the provider answers that the user no longer exists or is no longer allowed in, Dex ends all the sessions and tokens of the user of this provider. The data of local users is read again every time.

A provider that answers with a server error (HTTP 5xx) looks to Dex like a provider that is down. Once the stored data is older than `maxAge`, Dex answers the refresh of the tokens with 503 and keeps the tokens, and DexAuthenticator sends the user to the sign-in `idTokenTTL` after the first such error. Dex of the upstream project (dexidp/dex) and Deckhouse Dex before version 1.78 may answer the refresh of a deleted local or LDAP user this way. Behind such a provider, a deleted user sees an error page for about `idTokenTTL` and then the sign-in page.

A sign-in from a session needs fresh data too. Dex keeps no credentials of a user of an external provider to read the data in the background, so once the data is older than `maxAge`, such a user signs in to the next application with the password again. Every sign-in and every refresh of the tokens of an application that reads the provider renews the data: a user who keeps working in an application of a DexAuthenticator keeps the single sign-on. To keep it through longer breaks, raise `maxAge`.

- **settings.sessions.identityRefresh.gracePeriod**
  string

  How much longer than `maxAge` Dex serves the stored data when the provider does not answer.

  After this period, Dex refuses to refresh the tokens of the user until the provider answers.

  Set as a string of whole hours, minutes, and seconds, in this order: `30s`, `5m`, `1h`.

  Default: `0s`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.sessions.identityRefresh.maxAge**
  string

  Age of the stored data after which Dex reads it from the provider again.

  Defaults to `idTokenTTL`.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.sessions.rememberMeCheckedByDefault**
  boolean

  Whether the “Remember me” checkbox of the sign-in form is selected by default.

  Without “Remember me”, the session ends when the browser is closed.

  Default: `false`

- **settings.sessions.sso**
  string

  Sharing of the sign-in between applications.

  - `Platform`: The web interfaces of Deckhouse share the sign-in. These are the applications of the DexAuthenticator objects in the `d8-*` namespaces. User DexAuthenticator and DexClient objects do not share it.
  - `None`: Every application asks for the sign-in. A session only spares the password when the user signs in to the same application again.

  Default: `Platform`

  Allowed values: `Platform`, `None`

- **settings.sessions.validIfNotUsedFor**
  string

Time after which an unused session ends.

A session is used when the browser signs in to an application through it, and when an application signed in through it refreshes its tokens: an application of a DexAuthenticator object or of a DexClient with `refreshTokenLifetime: Session`. Such an application refreshes its tokens while it is open and sends requests; an application closed, or open in a browser of a sleeping computer, does not use the session. While the browser uses the session, it lasts up to `absoluteLifetime`.

With the default of a day, a session survives a night but not a weekend. For stricter requirements, set a shorter time, for example `1h`: the user then enters the password again after each break of an hour.

Must not be longer than `absoluteLifetime`. When only `absoluteLifetime` is set and is shorter than a day, an unused session ends after `absoluteLifetime`.

Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `2h30m`, `24h`.

Default: `24h`

Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted, it is determined automatically.

  Example:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string
- **settings.twoFactor**
  object

  Two-factor authentication (2FA) with a one-time code (TOTP) from an authenticator application, with a security key (WebAuthn), or with both, as `methods` and `require` set.

  The parameter replaces `staticUsers2FA` of the earlier versions of the settings, which covered local users only. The module converts `staticUsers2FA` to this parameter.

  - **settings.twoFactor.enabled**
    boolean

    Required value

    If set to `true`, the users of the providers listed in `providerTypes` will be required to use two-factor authentication (2FA) when logging in.

    Default: `false`

  - **settings.twoFactor.issuerName**
    string

    The issuer name for the two-factor authentication (2FA) tokens. This name is visible to users in the 2FA application (e.g., Google Authenticator) and in the dialog of the browser that registers a security key.

    Default: `Deckhouse`

- **settings.twoFactor.methods**
  array of strings

  Second factors of the users of the types in `providerTypes`, unless `providerTypeOverrides` sets others for a type:

  - `TOTP`: A one-time code from an authenticator application (for example, Google Authenticator).
  - `WebAuthn`: A security key (for example, YubiKey), or the Touch ID or Windows Hello of the device. The key confirms a sign-in only for the domain it is bound to, so a fake sign-in page on another domain gets no response it can reuse.

  With two methods, `require` sets how they combine.

  A user registers a method at the first sign-in that asks for it (and until then anyone who knows the password can register their own device). The secret or the keys the user had for a method that is no longer asked for stay until a `Reset2FA` operation, so asking for the method again brings them back.

  Default: `["TOTP"]`

  - **Element of the array**
    string

    Allowed values: `TOTP`, `WebAuthn`

- **settings.twoFactor.providerTypeOverrides**
  array of objects

  Second factors of the users of one type of provider, instead of `methods` and `require`. The type must be listed in `providerTypes`, and each type appears once.

  - **settings.twoFactor.providerTypeOverrides.methods**
    array of strings

    Required value

    Second factors of the users of this type, as `methods`.

    - **Element of the array**
      string

      Allowed values: `TOTP`, `WebAuthn`

  - **settings.twoFactor.providerTypeOverrides.providerType**
    string

    Required value

    Type of provider, one of `providerTypes`.

    Allowed values: `Local`, `LDAP`, `Crowd`

  - **settings.twoFactor.providerTypeOverrides.require**
    string

    How the two methods combine, as `require`. Required with two methods, and not allowed with one.

    Allowed values: `Any`, `All`

- **settings.twoFactor.providerTypes**
  array of strings

  Types of the providers whose users have to pass the second factor:

  - `Local`: The local users (User).
  - `LDAP`: The users of every DexProvider of the `LDAP` type.
  - `Crowd`: The users of every DexProvider of the `Crowd` type.

  A user enrolls the authenticator at the first sign-in after the second factor is turned on. Until then, anyone who knows the password can enroll their own device, so ask the users to sign in right after the switch.

  Adding a type does not end the refresh tokens or the Dex browser sessions issued before it. A refresh token keeps issuing tokens without the code until it expires, and a browser session created earlier passes without the code. After enabling 2FA, run a UserOperation of type `Logout` for the users of the providers of the listed types.

The second factor is not possible together with Basic Authentication to the Kubernetes API, which has no page for a code. A provider of a listed type cannot have `enableBasicAuth: true`. The module refuses a DexProvider that would have both, and refuses this setting while such a DexProvider exists.

The OIDC, SAML, GitHub, GitLab and Bitbucket Cloud providers are not on the list. The identity provider handles the second factor for them.

Default: `["Local"]`

- **Element of the array**
  string

  Allowed values: `Local`, `LDAP`, `Crowd`

- **settings.twoFactor.require**
  string

  How the two methods of `methods` combine.

  With `Any`, the user passes one of them. At the first sign-in the user chooses the method and registers it; later Dex asks for a registered method, the one used last first, and a user who has registered both can sign in another way. A user who has registered one method cannot register the other at a sign-in. To change the method, run a `Reset2FA` operation for the registered method or set `All`. With `All`, the user passes both, the code first and then the key. A user who has only one of them confirms the sign-in with it first and only then registers the other.

  Allowed values: `Any`, `All`

- **settings.twoFactor.webAuthn**
  object

  Parameters of the `WebAuthn` method.

  - **settings.twoFactor.webAuthn.maxCredentials**
    integer

    How many keys a user may register.

    The first key is registered at the first sign-in that asks for one. A user adds a further key on the page that confirms a sign-in with a key, after a registered key has confirmed it. A password alone never adds a key.

    Lowering the limit keeps the keys users have; a user with as many keys as the limit, or more, adds none.

    Default: `3`

    Allowed values: `1 <= X <= 10`

  - **settings.twoFactor.webAuthn.rpID**
    string

    Domain a registered key is bound to (the relying party ID). A key works only for this domain and its subdomains.

    The browser takes a domain that is the domain of the Dex address (`dex.<PUBLIC_DOMAIN>`) or its parent, down to a registrable domain (`example.com`), and refuses any other. By default, it is the domain of the Dex address, so a change of `publicDomainTemplate` makes every registered key useless, and every user has to register again. Set it to the stable parent domain to keep the keys through such a change.

    Setting or changing the parameter makes the keys registered before it useless as well.

    A user whose key has become useless can neither sign in nor register a new key, because Dex offers the registration only to an account that has no keys, and the old key cannot be used. Reset the second factor of such a user with a UserOperation of type `Reset2FA`; the user registers a key again at the next sign-in.

    Dex accepts `localhost` and a lowercase domain of two or more labels, such as `example.com` or `dex.example.com`. It refuses any other single label, a final label that is a number (`corp.123`, and a hexadecimal, binary or octal form such as `0x1`), and an IP address.

Pattern: `^(localhost|[a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?(\.[a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?)+)$`

Maximum length: `253`

Example:

```
rpID: example.com
```

> ⚠ The module works only with HTTPS, because OpenID Connect over HTTP is a serious security threat and the Kubernetes API server does not support OIDC over HTTP. With `https.mode` set to `Disabled`, the module is turned off.

Set `https.mode` to a value other than `Disabled` at the cluster level (`modules.https.mode`) or in the module. If you set the `https` section of the module, set `mode` in it too. Without `mode`, `Disabled` applies.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The user-authn module: Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

DexAuthenticator

Short names: dex-auth

Scope: Namespaced

- v1
- v2alpha1
- v1alpha1

Authentication through Dex for web applications in the namespace of the object.

A DexAuthenticator has no pods of its own. In the namespace of the object, the module creates the Service `<NAME>-dex-authenticator`, which the `auth-url` annotation of the application points at, and publishes the `https://<DOMAIN>/dex-authenticator` path through an Ingress or an HTTPRoute.

The `Served` condition in `status.conditions` shows whether the module serves the object.

> **Warning.** DexAuthenticator works only with HTTPS. The application Ingress must have TLS configured.

The name of a DexAuthenticator must be at most 63 characters long, start with a lowercase Latin letter and contain only lowercase Latin letters, digits and hyphens (RFC 1035).

Examples:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
  - everyone
  - admins
  allowedProviders:
  - local
  - corp-ldap
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
  additionalApplications:
  - domain: additional-app-name.kube.my-domain.com
    ingressSecretName: ingress-tls
    ingressClassName: nginx
    signOutURL: "/logout"
    whitelistSourceRanges:
    - 2.2.2.2/32
```

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-alb
  namespace: app-namespace
spec:
  applicationDomain: app.example.com
  gatewayAPI:
    applicationHTTPRouteListenerSetName: app-listeners
```

- **metadata**
  object
  - **metadata.name**
    string

    Maximum length: `63`

- **status**
  object

  State of the object, written by the module.

  - **status.conditions**
    array of objects

    Conditions of the object.

    The `Served` condition shows whether the dex-authenticator of the module serves the object on every domain set in the object. A new or changed object is served within a few minutes. If the status of the condition is `False`, `reason` is one of the following:

    - `WaitingForInstance`: The object is not served yet.
    - `HostConflict`: A domain of the object belongs to another DexAuthenticator. Change the domain, or the Ingress class if the domain is to be shared. The object is served on its other domains.
    - `InvalidRecord`: The object cannot be served as it is. `message` gives the reason.
    - `ClientNotCreated`: The module cannot register the application in Dex. `message` gives the reason.
    - `ForeignObject`: The namespace has a Service, Ingress or HTTPRoute with a name the

module needs. Rename or delete it.
- `ObjectsNotWritten`: The API server refused to create the Service or a route of the object, for example, because of an admission policy. `message` gives the answer of the API server.
- `TableFull`: The limit of served DexAuthenticator objects is reached in the cluster or in the namespace. `message` gives the details.
- **status.conditions.lastTransitionTime**
  string

  Time the status last changed.
- **status.conditions.message**
  string

  Details of the status.
- **status.conditions.observedGeneration**
  integer

  Generation of the object that the condition was set for.
- **status.conditions.reason**
  string

  Reason for the status, such as `HostConflict`.
- **status.conditions.status**
  string

  Required value

  Status of the condition.

  Allowed values: `True`, `False`, `Unknown`
- **status.conditions.type**
  string

  Required value

  Type of the condition, such as `Served`.

- **spec**
  object

  Required value

  - **spec.additionalApplications**
    array of objects

    A list of additional applications for which user authentication is required.

    - **spec.additionalApplications.domain**
      string

      Required value

      An application domain from which the user request will be redirected for authentication in Dex.

      > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`
    - **spec.additionalApplications.gatewayAPI**
      object

      A set of parameters to use in HTTPRoute object.

      > **Warning.** Only applicable in clusters with Gateway API support. Define the same ListenerSet object which is used to publish the application domain.

      - **spec.additionalApplications.gatewayAPI.httpRouteListenerSetName**

string

Required value

Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Maximum length: `253`

Example:

```
httpRouteListenerSetName: default-listener-set
```

- **spec.additionalApplications.gatewayAPI.httpRouteListenerSetNamespace**
  string

  Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

  If not set, the namespace of the `DexAuthenticator` is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `63`

  Example:

  ```
  httpRouteListenerSetNamespace: default
  ```

- **spec.additionalApplications.gatewayAPI.httpRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  httpRouteListenerSetSectionName: web-listener
  ```

- **spec.additionalApplications.ingressClassName**
  string

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Required unless `gatewayAPI` is specified. If omitted, no Ingress resources are created for this application. Both fields may be specified.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate (from the domain) used in the applications Ingress resource. The Secret must be in the same namespace as DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.additionalApplications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be

created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

The value must not contain line breaks.

Pattern: `^[^\r\n]*$`

Maximum length: `2048`

- **spec.additionalApplications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.allowedEmails**
  array of strings

  A list of emails of users are allowed to connect to the client.

  Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  By default, all users can connect.

  Values must not contain line breaks.

  Default: `All emails are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client.

  Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  By default, all groups can connect.

  Values must not contain line breaks.

  Default: `All groups are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `512`

- **spec.allowedProviders**
  array of strings

  Names of the DexProvider objects through which users can sign in to the application.

  Use `local` for the users of the built-in password database (User).

  A provider outside the list is not offered on the sign-in page. If the list holds one provider, users go straight to it. Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  A name that matches no DexProvider is accepted with a warning. Nobody can sign in through

it until the provider appears.

By default, all providers can be used.

Default: `All providers are allowed.`

- **Element of the array**
  string

  Pattern: `^[a-z0-9]([-a-z0-9.]*[a-z0-9])?$`

  Maximum length: `253`

- **spec.applicationDomain**
  string

  Required value

  An address for the application (DNS domain for Ingress resource) from which the user request will be redirected for authentication in Dex.

  > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationDomain: my-app.domain.com
  ```

- **spec.applicationIngressCertificateSecretName**
  string

  The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

  Example:

  ```
  applicationIngressCertificateSecretName: ingress-tls
  ```

- **spec.applicationIngressClassName**
  string

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Required unless `gatewayAPI` is specified. If omitted, no Ingress resources are created for this application. Both fields may be specified.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationIngressClassName: nginx
  ```

- **spec.gatewayAPI**
  object

  A set of parameters to use in HTTPRoute object.

  > **Warning.** Only applicable in clusters with Gateway API support. Define the same ListenerSet object which is used to publish the application domain.

  - **spec.gatewayAPI.applicationHTTPRouteListenerSetName**
    string

    Required value

Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Maximum length: `253`

Example:

```
applicationHTTPRouteListenerSetName: default-listenerset
```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**
  string

  Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

  If not set, the namespace of the `DexAuthenticator` is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `63`

  Example:

  ```
  applicationHTTPRouteListenerSetNamespace: default
  ```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  applicationHTTPRouteListenerSetSectionName: default-listener
  ```

- **spec.highAvailability**
  Deprecated
  boolean

  Ignored. Kept for compatibility.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `20s`, `2h30m10s`, `24h`.

  The value must be greater than `idTokenTTL`, otherwise the session lasts `idTokenTTL + 1s`. To sign users out sooner, decrease `idTokenTTL`.

  Default: `168h`

  Pattern: `^([0-9]{1,5}h)?([0-9]{1,7}m)?([0-9]{1,9}s)?$`

  Example:

  ```
  keepUsersLoggedInFor: 24h
  ```

- **spec.nodeSelector**
  Deprecated
  object

  Ignored. Kept for compatibility.

- **spec.podMetadata**
  Deprecated

object

Ignored. Kept for compatibility.

- **spec.podMetadata.annotations**
  object
- **spec.podMetadata.labels**
  object

- **spec.resources**
  Deprecated
  object

  Ignored. Kept for compatibility.

  - **spec.resources.limits**
    object
  - **spec.resources.redis**
    object
    - **spec.resources.redis.limits**
      object
    - **spec.resources.redis.requests**
      object
  - **spec.resources.requests**
    object
- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

  When enabled, the header carries a signed Dex ID token (`id_token`) containing the user's email, name, and group membership.

  > **Warning.** Enable it only if the application can process this header, and set up the route of the application as described in How to pass the user's login and groups to an application.

- **spec.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.tolerations**
  Deprecated
  array of objects

  Ignored. Kept for compatibility.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

    Allowed values: `Exists`, `Equal`

  - **spec.tolerations.tolerationSeconds**
    integer
  - **spec.tolerations.value**

string

- **spec.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Authentication through Dex for web applications in the namespace of the object.

A DexAuthenticator has no pods of its own. In the namespace of the object, the module creates the Service `<NAME>-dex-authenticator`, which the `auth-url` annotation of the application points at, and publishes the `https://<DOMAIN>/dex-authenticator` path through an Ingress or an HTTPRoute.

The `Served` condition in `status.conditions` shows whether the module serves the object.

> **Warning.** DexAuthenticator works only with HTTPS. The application Ingress must have TLS configured.

The name of a DexAuthenticator must be at most 63 characters long, start with a lowercase Latin letter and contain only lowercase Latin letters, digits and hyphens (RFC 1035).

- **metadata**
  object
  - **metadata.name**
    string

    Maximum length: `63`

- **status**
  object

  State of the object, written by the module.

  - **status.conditions**
    array of objects

    Conditions of the object.

    The `Served` condition shows whether the dex-authenticator of the module serves the object on every domain set in the object. A new or changed object is served within a few minutes. If the status of the condition is `False`, `reason` is one of the following:

    - `WaitingForInstance`: The object is not served yet.
    - `HostConflict`: A domain of the object belongs to another DexAuthenticator. Change the domain, or the Ingress class if the domain is to be shared. The object is served on its other domains.
    - `InvalidRecord`: The object cannot be served as it is. `message` gives the reason.
    - `ClientNotCreated`: The module cannot register the application in Dex. `message` gives the reason.
    - `ForeignObject`: The namespace has a Service, Ingress or HTTPRoute with a name the module needs. Rename or delete it.
    - `ObjectsNotWritten`: The API server refused to create the Service or a route of the object, for example, because of an admission policy. `message` gives the answer of the API server.
    - `TableFull`: The limit of served DexAuthenticator objects is reached in the cluster or in the namespace. `message` gives the details.
    - **status.conditions.lastTransitionTime**
      string

      Time the status last changed.

    - **status.conditions.message**
      string

Details of the status.

- **status.conditions.observedGeneration**
  integer

  Generation of the object that the condition was set for.

- **status.conditions.reason**
  string

  Reason for the status, such as `HostConflict`.

- **status.conditions.status**
  string

  Required value

  Status of the condition.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Type of the condition, such as `Served`.

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

    A list of emails of users are allowed to connect to the client.

    Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

    By default, all users can connect.

    Values must not contain line breaks.

    Default: `All users are allowed.`

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `320`

  - **spec.allowedGroups**
    array of strings

    A list of groups whose members are allowed to connect to the client.

    Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

    By default, all groups can connect.

    Values must not contain line breaks.

    Default: `All groups are allowed.`

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `512`

  - **spec.allowedProviders**
    array of strings

    Names of the DexProvider objects through which users can sign in to the application.

Use `local` for the users of the built-in password database (User).

A provider outside the list is not offered on the sign-in page. If the list holds one provider, users go straight to it. Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

A name that matches no DexProvider is accepted with a warning. Nobody can sign in through it until the provider appears.

By default, all providers can be used.

Default: `All providers are allowed.`

- **Element of the array**
  string

  Pattern: `^[a-z0-9]([-a-z0-9.]*[a-z0-9])?$`

  Maximum length: `253`

- **spec.applications**
  array of objects

  Required value

  A list of applications for which user authentication is required.

  - **spec.applications.domain**
    string

    Required value

    An application domain (for Ingress resources) from which the user request will be redirected for authentication in Dex.

    > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **spec.applications.gatewayAPI**
    object

    A set of parameters to use in HTTPRoute object.

    > **Warning.** Only applicable in clusters with Gateway API support. Define the same ListenerSet object which is used to publish the application domain.

    - **spec.applications.gatewayAPI.httpRouteListenerSetName**
      string

      Required value

      Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

      Maximum length: `253`

      Example:

      ```
      httpRouteListenerSetName: default-listener-set
      ```

    - **spec.applications.gatewayAPI.httpRouteListenerSetNamespace**
      string

      Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

      If not set, the namespace of the `DexAuthenticator` is used.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Maximum length: `63`

Example:

```
httpRouteListenerSetNamespace: default
```

- **spec.applications.gatewayAPI.httpRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  httpRouteListenerSetSectionName: web-listener
  ```

- **spec.applications.ingressClassName**
  string

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Required unless `gatewayAPI` is specified. If omitted, no Ingress resources are created for this application. Both fields may be specified.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.applications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.applications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.applications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, if not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.highAvailability**
  Deprecated
  boolean

  Ignored. Kept for compatibility.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `20s`, `2h30m10s`, `24h`.

  The value must be greater than `idTokenTTL`, otherwise the session lasts `idTokenTTL + 1s`. To sign users out sooner, decrease `idTokenTTL`.

  Default: `168h`

  Pattern: `^([0-9]{1,5}h)?([0-9]{1,7}m)?([0-9]{1,9}s)?$`

- **spec.nodeSelector**
  Deprecated
  object

  Ignored. Kept for compatibility.

- **spec.podMetadata**
  Deprecated
  object

  Ignored. Kept for compatibility.

  - **spec.podMetadata.annotations**
    object
  - **spec.podMetadata.labels**
    object
- **spec.resources**
  Deprecated
  object

  Ignored. Kept for compatibility.

  - **spec.resources.limits**
    object
  - **spec.resources.redis**
    object
    - **spec.resources.redis.limits**
      object
    - **spec.resources.redis.requests**
      object
  - **spec.resources.requests**
    object
- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

  When enabled, the header carries a signed Dex ID token (`id_token`) containing the user's email, name, and group membership.

  > **Warning.** Enable it only if the application can process this header, and set up the route of the application as described in How to pass the user's login and groups to an application.

- **spec.tolerations**
  Deprecated
  array of objects

  Ignored. Kept for compatibility.

  - **spec.tolerations.effect**

string

Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**
  string
- **spec.tolerations.operator**
  string

  Default: `Equal`

  Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer
- **spec.tolerations.value**
  string

Deprecated resource. Support for the resource might be removed in a later release.

Authentication through Dex for web applications in the namespace of the object.

A DexAuthenticator has no pods of its own. In the namespace of the object, the module creates the Service `<NAME>-dex-authenticator`, which the `auth-url` annotation of the application points at, and publishes the `https://<DOMAIN>/dex-authenticator` path through an Ingress or an HTTPRoute.

The `Served` condition in `status.conditions` shows whether the module serves the object.

> **i** **Warning.** DexAuthenticator works only with HTTPS. The application Ingress must have TLS configured.

The name of a DexAuthenticator must be at most 63 characters long, start with a lowercase Latin letter and contain only lowercase Latin letters, digits and hyphens (RFC 1035).

Examples:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
  - everyone
  - admins
  allowedProviders:
  - local
  - corp-ldap
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
  additionalApplications:
  - domain: additional-app-name.kube.my-domain.com
    ingressSecretName: ingress-tls
    ingressClassName: nginx
    signOutURL: "/logout"
    whitelistSourceRanges:
    - 2.2.2.2/32
```

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-alb
  namespace: app-namespace
spec:
  applicationDomain: app.example.com
  gatewayAPI:
    applicationHTTPRouteListenerSetName: app-listeners
```

- **metadata**
  object
  - **metadata.name**
    string

    Maximum length: `63`

- **status**
  object

  State of the object, written by the module.

  - **status.conditions**
    array of objects

    Conditions of the object.

    The `Served` condition shows whether the dex-authenticator of the module serves the object on every domain set in the object. A new or changed object is served within a few minutes. If the status of the condition is `False`, `reason` is one of the following:

    - `WaitingForInstance`: The object is not served yet.
    - `HostConflict`: A domain of the object belongs to another DexAuthenticator. Change the domain, or the Ingress class if the domain is to be shared. The object is served on its other domains.
    - `InvalidRecord`: The object cannot be served as it is. `message` gives the reason.
    - `ClientNotCreated`: The module cannot register the application in Dex. `message` gives the reason.
    - `ForeignObject`: The namespace has a Service, Ingress or HTTPRoute with a name the module needs. Rename or delete it.
    - `ObjectsNotWritten`: The API server refused to create the Service or a route of the object, for example, because of an admission policy. `message` gives the answer of the API server.
    - `TableFull`: The limit of served DexAuthenticator objects is reached in the cluster or in the namespace. `message` gives the details.
    - **status.conditions.lastTransitionTime**
      string

      Time the status last changed.

    - **status.conditions.message**
      string

      Details of the status.

    - **status.conditions.observedGeneration**
      integer

      Generation of the object that the condition was set for.

    - **status.conditions.reason**
      string

      Reason for the status, such as `HostConflict`.

    - **status.conditions.status**
      string

      Required value

      Status of the condition.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**

string

Required value

Type of the condition, such as `Served`.

- **spec**
  object

  Required value

  - **spec.additionalApplications**
    array of objects

    A list of additional applications for which user authentication is required.

    - **spec.additionalApplications.domain**
      string

      Required value

      An application domain from which the user request will be redirected for authentication in Dex.

      > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    - **spec.additionalApplications.gatewayAPI**
      object

      A set of parameters to use in HTTPRoute object.

      > **Warning.** Only applicable in clusters with Gateway API support. Define the same ListenerSet object which is used to publish the application domain.

      - **spec.additionalApplications.gatewayAPI.httpRouteListenerSetName**
        string

        Required value

        Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

        Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

        Maximum length: `253`

        Example:

        ```
        httpRouteListenerSetName: default-listener-set
        ```

      - **spec.additionalApplications.gatewayAPI.httpRouteListenerSetNamespace**
        string

        Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

        If not set, the namespace of the `DexAuthenticator` is used.

        Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

        Maximum length: `63`

        Example:

        ```
        httpRouteListenerSetNamespace: default
        ```

      - **spec.additionalApplications.gatewayAPI.httpRouteListenerSetSectionName**

string

Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Maximum length: `253`

Example:

```
httpRouteListenerSetSectionName: web-listener
```

- **spec.additionalApplications.ingressClassName**
  string

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Required unless `gatewayAPI` is specified. If omitted, no Ingress resources are created for this application. Both fields may be specified.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate (from the domain) used in the applications Ingress resource. The Secret must be in the same namespace as DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.additionalApplications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.additionalApplications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.allowedEmails**
  array of strings

  A list of emails of users are allowed to connect to the client.

  Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  By default, all users can connect.

Values must not contain line breaks.

Default: `All emails are allowed.`

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client.

  Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  By default, all groups can connect.

  Values must not contain line breaks.

  Default: `All groups are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `512`

- **spec.allowedProviders**
  array of strings

  Names of the DexProvider objects through which users can sign in to the application.

  Use `local` for the users of the built-in password database (User).

  A provider outside the list is not offered on the sign-in page. If the list holds one provider, users go straight to it. Changing the list ends the sessions of the users who no longer match it at their next token refresh, within `idTokenTTL`.

  A name that matches no DexProvider is accepted with a warning. Nobody can sign in through it until the provider appears.

  By default, all providers can be used.

  Default: `All providers are allowed.`

  - **Element of the array**
    string

    Pattern: `^[a-z0-9]([-a-z0-9.]*[a-z0-9])?$`

    Maximum length: `253`

- **spec.applicationDomain**
  string

  Required value

  An address for the application (DNS domain for Ingress resource) from which the user request will be redirected for authentication in Dex.

  > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationDomain: my-app.domain.com
  ```

- **spec.applicationIngressCertificateSecretName**
  string

The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Example:

```
applicationIngressCertificateSecretName: ingress-tls
```

- **spec.applicationIngressClassName**
  string

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Required unless `gatewayAPI` is specified. If omitted, no Ingress resources are created for this application. Both fields may be specified.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationIngressClassName: nginx
  ```

- **spec.gatewayAPI**
  object

  A set of parameters to use in HTTPRoute object.

  > **Warning.** Only applicable in clusters with Gateway API support. Define the same ListenerSet object which is used to publish the application domain.

  - **spec.gatewayAPI.applicationHTTPRouteListenerSetName**
    string

    Required value

    Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Maximum length: `253`

    Example:

    ```
    applicationHTTPRouteListenerSetName: default-listenerset
    ```

  - **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**
    string

    Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

    If not set, the namespace of the `DexAuthenticator` is used.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Maximum length: `63`

    Example:

    ```
    applicationHTTPRouteListenerSetNamespace: default
    ```

  - **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**
    string

    Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Maximum length: `253`

Example:

```
applicationHTTPRouteListenerSetSectionName: default-listener
```

- **spec.highAvailability**
  Deprecated
  boolean

  Ignored. Kept for compatibility.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string of whole hours, minutes, and seconds, in this order: `30m`, `20s`, `2h30m10s`, `24h`.

  The value must be greater than `idTokenTTL`, otherwise the session lasts `idTokenTTL + 1s`. To sign users out sooner, decrease `idTokenTTL`.

  Default: `168h`

  Pattern: `^([0-9]{1,5}h)?([0-9]{1,7}m)?([0-9]{1,9}s)?$`

  Example:

  ```
  keepUsersLoggedInFor: 24h
  ```

- **spec.nodeSelector**
  Deprecated
  object

  Ignored. Kept for compatibility.

- **spec.podMetadata**
  Deprecated
  object

  Ignored. Kept for compatibility.

  - **spec.podMetadata.annotations**
    object
  - **spec.podMetadata.labels**
    object
- **spec.resources**
  Deprecated
  object

  Ignored. Kept for compatibility.

  - **spec.resources.limits**
    object
  - **spec.resources.redis**
    object
    - **spec.resources.redis.limits**
      object
    - **spec.resources.redis.requests**
      object
  - **spec.resources.requests**
    object
- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

  When enabled, the header carries a signed Dex ID token (`id_token`) containing the user’s email, name, and group membership.

> **Warning.** Enable it only if the application can process this header, and set up the route of the application as described in How to pass the user's login and groups to an application.

- **spec.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.tolerations**
  Deprecated
  array of objects

  Ignored. Kept for compatibility.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

    Allowed values: `Exists`, `Equal`

  - **spec.tolerations.tolerationSeconds**
    integer
  - **spec.tolerations.value**
    string
- **spec.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

DexClient

Scope: Namespaced

- v1
- v1alpha1

Allows applications that support OIDC authentication to interact with Dex.

After the `DexClient` object appears in the cluster:

- Dex will register a client with a `dex-client-<NAME>@<NAMESPACE>` **clientID**, where `<NAME>` and `<NAMESPACE>` are `metadata.name` and `metadata.namespace` of the DexClient object, respectively.
- A `dex-client-<NAME>` Secret containing the client access password (**clientSecret**) will be created in the corresponding namespace (where `<NAME>` is `metadata.name` of the DexClient object).

Usage example…

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

    A list of emails of users are allowed to connect to the client.

    Checked on login and on every refresh-token request. Updating the list denies further refresh for sessions whose email is no longer listed.

    By default, all users can connect.

    Values must not contain line breaks.

    Default: `All users are allowed.`

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `320`

  - **spec.allowedGroups**
    array of strings

    A list of groups whose members are allowed to connect to the client.

    Checked on login and on every refresh-token request. Updating the list denies further refresh for sessions whose stored groups no longer intersect it.

    By default, all groups can connect.

    Values must not contain line breaks.

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `512`

  - **spec.allowedProviders**
    array of strings

    Names of the DexProvider objects through which users can sign in to the application.

    Use `local` for the users of the built-in password database (User).

    Checked on the sign-in page, on login, and on every refresh-token and token-exchange request. A provider outside the list is not offered on the sign-in page. If the list holds one provider, users go straight to it. Removing a provider from the list stops the refresh tokens that were issued through it.

    The DexClient can also exchange tokens through every TokenExchangeProvider that lists it in `spec.clients`, even when that provider is not in this list.

    A name that matches no DexProvider is accepted with a warning. Nobody can sign in through it until the provider appears.

    By default, all providers can be used.

    Default: `All providers are allowed.`

    - **Element of the array**
      string

      Pattern: `^[a-z0-9]([-a-z0-9.]*[a-z0-9])?$`

      Maximum length: `253`

  - **spec.backchannelLogoutURI**
    string

    Address Dex notifies when a browser session through which the user signed in to the

application ends (OpenID Connect Back-Channel Logout): the user signs out of the session; the provider of the user answers that the user no longer exists or is no longer allowed in; a UserOperation or the self-service of the web interface ends the session; the user is deleted. Dex sends the notification of a session ended outside Dex within seconds.

Works only with the browser sessions of Dex enabled. Dex sends the notification once, without retries: the application cannot rely on it to end its own session.

Must be an HTTPS address whose host is a name with a dot: Dex sends the notification from inside the cluster, so an IP address, `localhost`, a name without a dot or ending with one, and a name under `.svc` or `.cluster.local` are not accepted. A short name such as `<SERVICE>.<NAMESPACE>` cannot be told from an external name: use the full external name of the application.

Pattern: `^https://[^\r\n]*$`

Maximum length: `2048`

- **spec.postLogoutRedirectURIs**
  array of strings

  Addresses Dex may send the browser back to after the user signs out of Dex through the application (`post_logout_redirect_uri` of OpenID Connect RP-Initiated Logout).

  Works only with the browser sessions of Dex enabled.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `2048`

- **spec.redirectURIs**
  array of strings

  Array of URLs that Dex can redirect to after successful authentication.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `2048`

- **spec.refreshTokenLifetime**
  string

  How long the refresh tokens of the application live, with the browser sessions of Dex enabled.

  - `Standalone`: The tokens live until they expire, whatever happens to the session of the browser. An application that requests offline access then gets a refresh token only after an interactive sign-in.
  - `Session`: The tokens end when the user signs out of the session of Dex they were issued through. Such an application can sign the user in from the session of the browser.

  Without the parameter, the tokens are `Standalone`.

  Allowed values: `Standalone`, `Session`

- **spec.secretMetadata**
  object

  Labels and annotations to add to the `dex-client-<NAME>` Secret.

  - **spec.secretMetadata.annotations**
    object

    Annotations of the Secret.

  - **spec.secretMetadata.labels**
    object

    Labels of the Secret.

- **spec.trustedPeers**
  array of strings

  OAuth2 client IDs that allowed cross authentication with the current client.

  A peer can ask for a token whose audience is this client through a provider the peer's own `allowedProviders` allows. This client's `allowedProviders` is not checked for that token, and neither are `allowedEmails` and `allowedGroups`.

  Values must not contain line breaks.

  Details...

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `256`

Deprecated resource. Support for the resource might be removed in a later release.

Allows applications that support OIDC authentication to interact with Dex.

After the `DexClient` object appears in the cluster:

- Dex will register a client with a `dex-client-<NAME>@<NAMESPACE>` **clientID**, where `<NAME>` and `<NAMESPACE>` are `metadata.name` and `metadata.namespace` of the DexClient object, respectively.
- A `dex-client-<NAME>` Secret containing the client access password (**clientSecret**) will be created in the corresponding namespace (where `<NAME>` is `metadata.name` of the DexClient object).

Usage example...

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

    A list of emails of users are allowed to connect to the client.

    Checked on login and on every refresh-token request. Updating the list denies further refresh for sessions whose email is no longer listed.

    By default, all users can connect.

    Values must not contain line breaks.

    Default: `All users are allowed.`

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `320`

  - **spec.allowedGroups**
    array of strings

    A list of groups whose members are allowed to connect to the client.

    Checked on login and on every refresh-token request. Updating the list denies further refresh for sessions whose stored groups no longer intersect it.

    By default, all groups can connect.

    Values must not contain line breaks.

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `512`

  - **spec.allowedProviders**
    array of strings

Names of the DexProvider objects through which users can sign in to the application.

Use `local` for the users of the built-in password database (User).

Checked on the sign-in page, on login, and on every refresh-token and token-exchange request. A provider outside the list is not offered on the sign-in page. If the list holds one provider, users go straight to it. Removing a provider from the list stops the refresh tokens that were issued through it.

The DexClient can also exchange tokens through every TokenExchangeProvider that lists it in `spec.clients`, even when that provider is not in this list.

A name that matches no DexProvider is accepted with a warning. Nobody can sign in through it until the provider appears.

By default, all providers can be used.

Default: `All providers are allowed.`

- **Element of the array**
  string

  Pattern: `^[a-z0-9]([-a-z0-9.]*[a-z0-9])?$`

  Maximum length: `253`

- **spec.backchannelLogoutURI**
  string

  Address Dex notifies when a browser session through which the user signed in to the application ends (OpenID Connect Back-Channel Logout): the user signs out of the session; the provider of the user answers that the user no longer exists or is no longer allowed in; a UserOperation or the self-service of the web interface ends the session; the user is deleted. Dex sends the notification of a session ended outside Dex within seconds.

  Works only with the browser sessions of Dex enabled. Dex sends the notification once, without retries: the application cannot rely on it to end its own session.

  Must be an HTTPS address whose host is a name with a dot: Dex sends the notification from inside the cluster, so an IP address, `localhost`, a name without a dot or ending with one, and a name under `.svc` or `.cluster.local` are not accepted. A short name such as `<SERVICE>.<NAMESPACE>` cannot be told from an external name: use the full external name of the application.

  Pattern: `^https://[^\r\n]*$`

  Maximum length: `2048`

- **spec.postLogoutRedirectURIs**
  array of strings

  Addresses Dex may send the browser back to after the user signs out of Dex through the application (`post_logout_redirect_uri` of OpenID Connect RP-Initiated Logout).

  Works only with the browser sessions of Dex enabled.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `2048`

- **spec.redirectURIs**
  array of strings

  Array of URLs that Dex can redirect to after successful authentication.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `2048`

- **spec.refreshTokenLifetime**
  string

  How long the refresh tokens of the application live, with the browser sessions of Dex enabled.

  - `Standalone`: The tokens live until they expire, whatever happens to the session of the browser. An application that requests offline access then gets a refresh token only after an interactive sign-in.
  - `Session`: The tokens end when the user signs out of the session of Dex they were issued through. Such an application can sign the user in from the session of the browser.

  Without the parameter, the tokens are `Standalone`.

  Allowed values: `Standalone`, `Session`

- **spec.secretMetadata**
  object

  Labels and annotations to add to the `dex-client-<NAME>` Secret.

  - **spec.secretMetadata.annotations**
    object

    Annotations of the Secret.

  - **spec.secretMetadata.labels**
    object

    Labels of the Secret.

- **spec.trustedPeers**
  array of strings

  OAuth2 client IDs that allowed cross authentication with the current client.

  A peer can ask for a token whose audience is this client through a provider the peer's own `allowedProviders` allows. This client's `allowedProviders` is not checked for that token, and neither are `allowedEmails` and `allowedGroups`.

  Values must not contain line breaks.

  Details...

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `256`

DexProviderCheck

Scope: Cluster
Version: v1

Describes a one-time connectivity check for a DexProvider.

The check verifies that the provider exists, is enabled, Dex is reachable, and the provider endpoint is reachable. It does not perform a full user authentication flow.

- **spec**
  object

  Required value

  - **spec.initiatorType**
    string

    Indicates who initiated the check. Possible values:

    - `Admin`: By an administrator.
    - `System`: Automatically by the system.

    Allowed values: `Admin`, `System`

  - **spec.providerName**
    string

    Required value

Name of the DexProvider resource to check.

Example:

```
providerName: my-oidc
```

- **status**
  object
  - **status.checks**
    array of objects

    Detailed results for individual check steps.

    - **status.checks.message**
      string

      Optional check step message.

    - **status.checks.name**
      string

      Required value

      Check step name.

    - **status.checks.status**
      string

      Required value

      Check step status. Possible values:

      - `Succeeded`: Step completed successfully.
      - `Failed`: Step failed.
      - `Skipped`: Step was not applicable and was skipped.
      - `Warning`: Step completed, but a setting needs attention (for example, disabled TLS verification or a soon-to-expire certificate). It does not fail the overall check.

      Allowed values: `Succeeded`, `Failed`, `Skipped`, `Warning`

  - **status.completedAt**
    string

    Time when the check completed.

  - **status.message**
    string

    Optional summary of the result or failure reason.

  - **status.observedDexProviderGeneration**
    integer

    DexProvider generation observed by the check.

  - **status.phase**
    string

    Current phase of the check. Possible values:

    - `Pending`: The check has been picked up and is awaiting or in progress.
    - `Succeeded`: Connectivity check completed successfully.
    - `Failed`: Connectivity check failed.

    Allowed values: `Pending`, `Succeeded`, `Failed`

DexProvider

Scope: Cluster

- v1
- v1alpha1

Defines the configuration for connecting a third-party provider.

With it, you can flexibly configure the integration of the account directory with Kubernetes.

Usage example…

Warning. It is forbidden to use users and groups with the `system:` prefix. Authentication attempts by such users or members of such groups will be rejected, and a corresponding warning will appear in the `kube-apiserver` logs.

- **spec**
  object

  Required value

  - **spec.allowedIdentities**
    object

    Limits the identities this provider is allowed to assert, regardless of the provider type.

    Dex refuses to sign in a user whose email is outside `emails` and `emailDomains` when either is set, and narrows the `groups` claim to the intersection with `groups` when it is set (an empty intersection refuses the sign-in). Each list is optional; an omitted or empty list does not limit that attribute.

    Dex also checks the limits at every refresh-token request and at Basic Authentication of an LDAP provider. It revokes a refresh token outside the limits, and the user has to sign in again. For a Crowd or OIDC provider, basic-auth-proxy checks the credentials with the provider directly, so the limits do not apply to Basic Authentication.

    The limits also decide who may create or change the provider. A provider that can assert an email or a group that already carries roles in the cluster may be written only by a requester who is allowed to assign those roles. Without an email limit the provider can assert any email, including the ones that hold `SuperAdmin`, so writing it requires `SuperAdmin`. With limits on both attributes only the roles granted to the listed identities count. A Crowd or OIDC provider with `enableBasicAuth: true` counts as a provider without an email limit, because Basic Authentication bypasses the limits.

    - **spec.allowedIdentities.emailDomains**
      array of strings

      Email domains this provider may assert, without the `@`.

      Matched exactly against the part of the email after the last `@`; subdomains must be listed explicitly.

      - **Element of the array**
        string
    - **spec.allowedIdentities.emails**
      array of strings

      Exact email addresses this provider may assert.

      Compared case-insensitively.

      - **Element of the array**
        string
    - **spec.allowedIdentities.groups**
      array of strings

      Group names this provider may assert, with the `groupPrefix` in front when it is set.

      Applied on top of the provider-specific group filters (`allowedGroups`, `groups`, `teams`, `orgs[].teams`), if any.

      - **Element of the array**
        string

        Length: `1..256`

  - **spec.bitbucketCloud**
    object

    Parameters of the Bitbucket Cloud (intended for the `type: BitbucketCloud`).

    - **spec.bitbucketCloud.clientID**
      string

      Required value

      Team application ID from BitbucketCloud (Key).

    - **spec.bitbucketCloud.clientSecret**
      string

Required value

Team application secret key from BitbucketCloud.

If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.bitbucketCloud.includeTeamGroups**
  boolean

  Optional parameter to include team groups.

  If enabled, the groups claim of Dex id_token will look like this:

  ```
  ["my_team", "my_team/administrators", "my_team/members"]
  ```

  Default: `false`

- **spec.bitbucketCloud.teams**
  array of strings

  A list of allowed Bitbucket Cloud teams (filter).

  The user token will contain a set intersection of Bitbucket Cloud teams and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain the user teams in the `groups` claim (similar to other providers).

- **spec.crowd**
  object

  Parameters of the Crowd (intended for the `type: Crowd`).

  - **spec.crowd.baseURL**
    string

    Required value

    Base part of Atlassian Crowd URL.

    Example:

    ```
    baseURL: https://crowd.example.com/crowd
    ```

  - **spec.crowd.clientID**
    string

    Required value

    Application ID from Atlassian Crowd (Application Name).

  - **spec.crowd.clientSecret**
    string

    Required value

    Application secret key from Atlassian Crowd (Password).

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.crowd.enableBasicAuth**
    boolean

    Enables Basic Authentication to the Kubernetes API server with the Crowd user’s username and password.

    Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

    The parameter cannot be turned on while `twoFactor.providerTypes` includes `Crowd`, because Basic Authentication has no page for the second factor.

    Basic Authentication bypasses `spec.allowedIdentities`. Therefore only a `SuperAdmin` can create a provider with this parameter enabled or enable the parameter

in an existing provider.

A successful check is cached together with the user's groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.crowd.groups**
  array of strings

  A list of allowed Crowd groups (filter).

  The user token will contain a set intersection of Crowd groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain all Crowd groups if the parameter is not set.

- **spec.crowd.usernamePrompt**
  string

  Prompt for username field.

  Default: `Crowd username`

- **spec.displayName**
  string

  Required value

  The provider name to show on the authentication provider selection page. The selection page will not be displayed if there is only one provider configured.

- **spec.enabled**
  boolean

  Whether the provider is enabled.

  A disabled provider is not shown on the sign-in page, and users cannot sign in through it. The DexProvider and its settings are kept.

  Default: `true`

- **spec.github**
  object

  Parameters of the GitHub provider (intended for the `type: Github` case only).

  - **spec.github.clientID**
    string

    Required value

    Organization application ID from GitHub.

  - **spec.github.clientSecret**
    string

    Required value

    Organization application secret key from GitHub.

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.github.orgs**
    array of objects

    Filter for user organizations. ID token will contain only organizations from this list. If the user is not in any organization from this list, an authorization will fail.

    By default, all organizations allowed.

    - **spec.github.orgs.name**
      string

      Required value

      Name of organization.

    - **spec.github.orgs.teams**
      array of strings

A list of allowed GitHub teams (filter).

The user token will contain a set intersection of teams from GitHub and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

The user token will contain all GitHub teams if the parameter is not set.

- **spec.github.teamNameField**
  string

  As an example, group claims for member of ‘Site Reliability Engineers’ in Acme organization would yield:

  - [‘acme:Site Reliability Engineers’] for ‘Name’
  - [‘acme:site-reliability-engineers’] for ‘Slug’
  - [‘acme:Site Reliability Engineers’, ‘acme:site-reliability-engineers’] for ‘Both’

  Default: `Name`

  Allowed values: `Name`, `Slug`, `Both`

- **spec.github.useLoginAsID**
  boolean

  Flag which will switch from using the internal GitHub id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.gitlab**
  object

  Parameters of the GitLab provider (intended for the `type: Gitlab` case only).

  - **spec.gitlab.baseURL**
    string

    Base part of GitLab URL.

    Example:

    ```
    baseURL: https://gitlab.example.com
    ```

  - **spec.gitlab.clientID**
    string

    Required value

    Application ID from GitLab.

  - **spec.gitlab.clientSecret**
    string

    Required value

    Application secret key from GitLab.

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.gitlab.groups**
    array of strings

    A list (filter) of allowed GitLab groups (**group paths** and not names).

    The user token will contain a set intersection of GitLab groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

    The user token will contain all GitLab groups if the parameter is not set;

  - **spec.gitlab.rootCAData**
    string

    A CA chain to validate the provider in PEM format.

    Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.gitlab.useLoginAsID**
  boolean

  Flag to switch from using the internal GitLab id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.groupPrefix**
  string

  Prefix that Dex puts in front of every group the provider asserts, whatever the type of the provider.

  With a prefix, the groups of this provider are named `<PREFIX><GROUP>` in the tokens and in the rules of the cluster, for example `corp-ldap:admins`. The prefix keeps them apart from the groups of another provider only when every provider that can emit such names has its own prefix.

  The prefix must end with a colon, as in `corp-ldap:`. It must differ from the prefix of every other DexProvider and of every TokenExchangeProvider group, and neither of two prefixes may start with the other. The admission webhooks refuse such a prefix.

  Every name in `allowedIdentities.groups` must start with the prefix, otherwise the API refuses the manifest, because Dex adds the prefix before it compares. The group settings of the provider type (for example, `oidc.allowedGroups`, `gitlab.groups` and `ldap.groupSearch`) take the names of the identity provider, without the prefix.

  The prefix cannot start with `system:`, which the Kubernetes API server reserves, or with `peering:`, which is reserved for the ServiceAccounts of the `d8-peering` namespace. It cannot be set together with `enableBasicAuth` of an `OIDC` or `Crowd` provider, because basic-auth-proxy checks the credentials with such a provider directly and passes its groups as they are.

  Setting or changing the prefix renames the groups of the provider, and the rules written for the old names stop applying to its users. Rewrite the rules before you set the prefix, as described in Prefix for the groups of a provider.

  By default, no prefix.

  Pattern: `^[A-Za-z0-9][A-Za-z0-9._:/-]*:$`

  Length: `2..64`

  Example:

  ```
  groupPrefix: 'corp-ldap:'
  ```

- **spec.ldap**
  object

  Parameters of the LDAP.

  - **spec.ldap.bindDN**
    string

    The DN for an application service account. The connector uses these credentials to search for users and groups. Not required if the LDAP server provides access for anonymous auth.

    Example:

    ```
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    ```

  - **spec.ldap.bindPW**
    string

    Password of the service account set in `bindDN`.

If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

Example:

```
bindPW: password
```

- **spec.ldap.enableBasicAuth**
  boolean

  Enables Basic Authentication to the Kubernetes API server with the LDAP user's username and password.

  Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

  The parameter cannot be turned on while `twoFactor.providerTypes` includes `LDAP`, because Basic Authentication has no page for the second factor.

  A successful check is cached together with the user's groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.ldap.groupSearch**
  object

  Group search queries for groups given a user entry. Details

  - **spec.ldap.groupSearch.baseDN**
    string

    Required value

    BaseDN to start the search from.

    Example:

    ```
    baseDN: cn=users,dc=example,dc=com
    ```

  - **spec.ldap.groupSearch.filter**
    string

    Optional filter to apply when searching the directory.

    Example:

    ```
    filter: "(objectClass=person)"
    ```

  - **spec.ldap.groupSearch.nameAttr**
    string

    Required value

    Represents group name.

    Example:

    ```
    nameAttr: name
    ```

  - **spec.ldap.groupSearch.userMatchers**
    array of objects

    Required value

    Following list contains field pairs that are used to match a user to a group. It adds a requirement to the filter that an attribute in the group must match the user's attribute value.

    - **spec.ldap.groupSearch.userMatchers.groupAttr**
      string

      Required value

      The name of the attribute that stores the group member names.

Example:

```
groupAttr: member
```

- **spec.ldap.groupSearch.userMatchers.recursionGroupAttr**
  string

  Name of the group attribute that stores the DNs of its members.

  When it is set, Dex also looks for the groups that contain the found groups, level by level, and adds them to the groups of the user. Use it for nested groups. The value is usually the same as `groupAttr`, for example `member`.

  Dex makes one more search for every group of every level at each sign-in and each refresh of the tokens, so keep `baseDN` and `filter` of `groupSearch` narrow.

  Example:

  ```
  recursionGroupAttr: member
  ```

- **spec.ldap.groupSearch.userMatchers.userAttr**
  string

  Required value

  The name of the attribute that stores the user name.

  Example:

  ```
  userAttr: uid
  ```

- **spec.ldap.host**
  string

  Required value

  Host and optional port of the LDAP server in the form “host:port”. If the port is not supplied, it will be guessed based on “insecureNoSSL”, and “startTLS” flags. 389 for insecure or StartTLS connections, 636 otherwise.

  Example:

  ```
  host: ldap.example.com:636
  ```

- **spec.ldap.insecureNoSSL**
  boolean

  Following field is required if the LDAP host is not using TLS (port 389). This option inherently leaks passwords to anyone on the same network as Dex.

  Mutually exclusive with `startTLS`, `insecureSkipVerify`, and `rootCAData`: when `insecureNoSSL` is true, TLS is disabled and those settings have no effect.

  Default: `false`

- **spec.ldap.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Has no effect when `insecureNoSSL` is true.

  Default: `false`

- **spec.ldap.kerberos**
  object

  Optional Kerberos (SPNEGO) SSO settings for LDAP.

  - **spec.ldap.kerberos.enabled**
    boolean

    Enable Kerberos SPNEGO flow for this LDAP provider.

Default: `false`

- **spec.ldap.kerberos.expectedRealm**
  string

  Optional realm to require (case-insensitive).

- **spec.ldap.kerberos.fallbackToPassword**
  boolean

  If `true`, in the absence of/incorrect `Authorization: Negotiate` header, a login/password form will be displayed instead of a 401 response with `WWW-Authenticate: Negotiate`.

  Default: `false`

- **spec.ldap.kerberos.keytabSecretName**
  string

  Name of the Secret in the `d8-user-authn` namespace containing the `krb5.keytab` key to mount for Dex.

  Required when `enabled` is `true`.

- **spec.ldap.kerberos.krb5ConfConfigMapName**
  string

  Not used. Dex validates Kerberos tickets with the keytab only, and the module does not mount this ConfigMap.

- **spec.ldap.kerberos.krb5ConfSecretName**
  string

  Not used. Dex validates Kerberos tickets with the keytab only, and the module does not mount this Secret.

- **spec.ldap.kerberos.usernameFromPrincipal**
  string

  How to map Kerberos principal to LDAP username.

  Default: `Localpart`

  Allowed values: `Localpart`, `SAMAccountName`, `UserPrincipalName`

- **spec.ldap.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Has no effect when `insecureNoSSL` is true.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.ldap.startTLS**
  boolean

  When connecting to the server, connect using the ldap:// protocol then issue a StartTLS command. If unspecified, connections will use the ldaps:// protocol.

  Mutually exclusive with `insecureNoSSL`.

  Default: `false`

- **spec.ldap.userSearch**
  object

  Required value

  User search maps a username and password entered by a user to a LDAP entry.
  Details...

  - **spec.ldap.userSearch.baseDN**

string

Required value

BaseDN to start the search from.

Example:

```
baseDN: cn=users,dc=example,dc=com
```

- **spec.ldap.userSearch.emailAttr**
  string

  Required value

  LDAP attribute that will be matched to Dex user email entry. When an email address is not available, use another value unique to the user, like uid.

  Example:

  ```
  emailAttr: mail
  ```

- **spec.ldap.userSearch.filter**
  string

  Optional filter to apply when searching the directory.

  Example:

  ```
  filter: "(objectClass=person)"
  ```

- **spec.ldap.userSearch.idAttr**
  string

  Required value

  LDAP attribute that will be matched to Dex user id entry.

  The binary GUID attributes of Active Directory, `objectGUID` and `mS-DS-ConsistencyGuid`, are supported: the user ID is then the GUID in its string form, for example `7ed41c2b-e283-4b10-9cb8-a6e7d1f39a4c`.

  Example:

  ```
  idAttr: uid
  ```

- **spec.ldap.userSearch.nameAttr**
  string

  LDAP attribute that will be matched to Dex user name entry. No default value provided.

  Example:

  ```
  nameAttr: name
  ```

- **spec.ldap.userSearch.username**
  string

  Username attribute used for comparing user entries. This will be translated and combined with the other filter as “(=)".

  Example:

  ```
  username: uid
  ```

- **spec.ldap.userSearch.usernames**
  array of strings

  Attributes that the entered username is compared with, instead of `username`.

  The search filter has the form `(|(<ATTR_1>=<USERNAME>)(<ATTR_2>=<USERNAME>))`, so a user can sign in, for example, with a login or an email. A sign-

in is refused if the username matches more than one entry.

The Kerberos principal mapped to a username (after `usernameFromPrincipal`) is compared with every attribute in the list the same way. List only attributes whose values are unique in the directory and consistent with `usernameFromPrincipal`. If a value of another entry equals the principal, SSO stops working for that user. A principal that matches none of the attributes of its own entry is matched to the entry of another user.

Set either `username` or `usernames`.

Example:

```
usernames:
- uid
- mail
```

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Length: `1..256`

- **spec.ldap.usernamePrompt**
  string

  The attribute to display in the provided password prompt. If unset, will display “LDAP Username”.

  Default: `LDAP username`

  Example:

  ```
  usernamePrompt: SSO Username
  ```

- **spec.oidc**
  object

  Parameters of the OIDC (intended for the `type: OIDC`).

  - **spec.oidc.allowedGroups**
    array of strings

    A list of groups the user must belong to in order to authenticate successfully.

    If none of the specified groups match the user’s groups, authentication will be denied.

    If the parameter is omitted, no group-based filtering will be applied.

  - **spec.oidc.basicAuthUnsupported**
    boolean

    Use POST requests to interact with the provider instead of including the token in the Basic Authorization header. Generally, Dex automatically determines the type of request to make, while in some cases enabling this parameter can help.

    Default: `false`

  - **spec.oidc.claimMapping**
    object

    Some providers return non-standard claims (eg. mail). Claim mappings are hints for Dex how to map claims to standard OIDC claims.

    Dex can only map a non-standard claim to a standard one if it’s not included in the id_token returned by OIDC provider.

    - **spec.oidc.claimMapping.email**
      string

      The claim to use as the user email.

      Default: `email`

    - **spec.oidc.claimMapping.groups**
      string

The claim to use as the user groups.

Default: `groups`

- **spec.oidc.claimMapping.preferred_username**
  string

  The claim to use as the user preferred username.

  Default: `preferred_username`

- **spec.oidc.claimMappingOverride**
  boolean

  If enabled, the claim mapping will override the standard OIDC claims.

  By default, the claim mapping will be used only if the standard OIDC claims are not present, e.g., if there is no `email` claim in the id_token, the `claimMapping.email` will be used.

  Default: `false`

- **spec.oidc.clientID**
  string

  Required value

  OIDC issuer application ID.

- **spec.oidc.clientSecret**
  string

  Required value

  OIDC issuer application secret key.

  If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.oidc.enableBasicAuth**
  boolean

  Enables Basic Authentication to the Kubernetes API server with the OIDC user’s username and password.

  Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

  Basic Authentication bypasses `spec.allowedIdentities`. Therefore only a `SuperAdmin` can create a provider with this parameter enabled or enable the parameter in an existing provider.

  A successful check is cached together with the user’s groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.oidc.getUserInfo**
  boolean

  Request additional info about the authenticated user.

  See UserInfo endpoint.

  Default: `false`

- **spec.oidc.insecureSkipEmailVerified**
  boolean

  Allow authentication for clients without verified email address.

  Default: `false`

- **spec.oidc.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

Default: `false`

- **spec.oidc.issuer**
  string

  Required value

  Canonical URL of the provider, also used for configuration discovery. This value MUST match the value returned in the provider config discovery.

  Example:

  ```
  issuer: https://accounts.google.com
  ```

- **spec.oidc.promptType**
  string

  Determines if the Issuer should ask for confirmation and provide hints during the authentication process.

  By default, the confirmation will be requested on the first authentication. Possible values may vary depending on the Issuer.

  Default: `consent`

- **spec.oidc.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.oidc.scopes**
  array of strings

  List of additional scopes to request in token response.

  A scope is a single token and must not contain whitespace: the OAuth2 protocol passes scopes as a space-delimited list.

  Default: `["openid","profile","email","groups","offline_access"]`

  - **Element of the array**
    string

    Pattern: `^\S+$`

- **spec.oidc.upstreamLogout**
  boolean

  Signs the user out of the provider too when the user signs out of Dex.

  Works only with the browser sessions of Dex enabled and only when the provider advertises an `end_session_endpoint`.

  By default, signing out of Dex leaves the session of the provider alone: signing out of one application does not sign the user out of everything else that signs in through the provider. The provider may then sign the user in again without a password. To ask for the password at every sign-in, set `promptType: login`.

  Default: `false`

- **spec.oidc.userIDKey**
  string

  The claim to use as the user id.

  Default: `sub`

- **spec.oidc.userNameKey**
  string

The claim to use as the user name.

Default: `name`

- **spec.saml**
  object

  Parameters of a SAML provider (intended for the `type: SAML`).

  - **spec.saml.allowedGroups**
    array of strings

    If set, only members of at least one of the listed groups will be allowed to log in.

  - **spec.saml.emailAttr**
    string

    SAML attribute to map to the email claim.

    Default: `email`

  - **spec.saml.entityIssuer**
    string

    Issuer value that Dex sends in the SAML AuthnRequest (the entity ID of Dex as a service provider). Dex also expects this value in the audience of the SAML assertion.

    If not set, the request carries no Issuer, and Dex expects the callback URL in the audience.

  - **spec.saml.filterGroups**
    boolean

    If `true`, the connector will filter the groups claim to only include groups from the `allowedGroups` list.

    Default: `false`

  - **spec.saml.groupsAttr**
    string

    SAML attribute to map to the groups claim.

  - **spec.saml.groupsDelim**
    string

    Delimiter to split the groups attribute value if it is a single string.

  - **spec.saml.insecureSkipSignatureValidation**
    boolean

    Skip signature validation of SAML responses. This is insecure and should only be used for testing.

    Default: `false`

  - **spec.saml.nameIDPolicyFormat**
    string

    The NameIDPolicy format to request from the IdP.

    If not set, Dex requests `persistent`.

    A UserOperation `Lock` with `spec.target` applies to the user ID, which Dex takes from the NameID. Use `persistent` or `emailAddress` for such a lock to work. With `transient`, the IdP returns a new NameID at every sign-in, and a lock does not apply to the next sign-in of the user.

    Allowed values: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`

  - **spec.saml.redirectURI**
    string

    Callback URI.

    Overrides the default callback URL. Should be set to `https://<dex-domain>/callback`.

  - **spec.saml.rootCAData**

string

A CA chain to validate the provider in PEM format.

Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.saml.ssoIssuer**
  string

  Issuer value expected in the SAML response from the IdP.

  If not set, the Issuer of the response is not checked.

- **spec.saml.ssoURL**
  string

  Required value

  SSO URL used for POST value.

  Example:

  ```
  ssoURL: https://idp.example.com/saml/sso
  ```

- **spec.saml.usernameAttr**
  string

  SAML attribute to map to the username claim.

  Default: `name`

- **spec.tokenExchange**
  object

  Settings of token exchange (RFC 8693) through the provider. Dex exchanges an ID token of the provider's issuer for its own token. Only `OIDC` and `Gitlab` providers can exchange tokens.

  The provider's connector does not check the audience of the exchanged token, so any application of the issuer could get a cluster token. Exchange tokens through a TokenExchangeProvider instead. The field is temporary and will be removed after the next LTS release, together with token exchange through a DexProvider.

  - **spec.tokenExchange.enabled**
    boolean

    Whether the provider exchanges tokens. Absent means `false`.

    Exchange can only be turned off. An administrator cannot set `true`, either when creating the provider or when changing it. During the upgrade from a release without this field, `user-authn` sets it once on every existing `OIDC` and `Gitlab` provider. It sets `true` on all of them if at least one DexClient has the `dexclient.deckhouse.io/allow-access-to-kubernetes: "true"` annotation, and `false` on all of them otherwise. A DexAuthenticator with access to Kubernetes does not count. Its oauth2-proxy never exchanges tokens, and exchange with its client secret, taken from the Secret in the namespace of the application, is not kept.

- **spec.type**
  string

  Required value

  Type of authentication provider.

  Allowed values: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Deprecated resource. Support for the resource might be removed in a later release.

Defines the configuration for connecting a third-party provider.

With it, you can flexibly configure the integration of the account directory with Kubernetes.

Usage example…

- **spec**
  object

  Required value

  - **spec.allowedIdentities**
    object

    Limits the identities this provider is allowed to assert, regardless of the provider type.

    Dex refuses to sign in a user whose email is outside `emails` and `emailDomains` when either is set, and narrows the `groups` claim to the intersection with `groups` when it is set (an empty intersection refuses the sign-in). Each list is optional; an omitted or empty list does not limit that attribute.

    Dex also checks the limits at every refresh-token request and at Basic Authentication of an LDAP provider. It revokes a refresh token outside the limits, and the user has to sign in again. For a Crowd or OIDC provider, basic-auth-proxy checks the credentials with the provider directly, so the limits do not apply to Basic Authentication.

    The limits also decide who may create or change the provider. A provider that can assert an email or a group that already carries roles in the cluster may be written only by a requester who is allowed to assign those roles. Without an email limit the provider can assert any email, including the ones that hold `SuperAdmin`, so writing it requires `SuperAdmin`. With limits on both attributes only the roles granted to the listed identities count. A Crowd or OIDC provider with `enableBasicAuth: true` counts as a provider without an email limit, because Basic Authentication bypasses the limits.

    - **spec.allowedIdentities.emailDomains**
      array of strings

      Email domains this provider may assert, without the `@`.

      Matched exactly against the part of the email after the last `@`; subdomains must be listed explicitly.

      - **Element of the array**
        string
    - **spec.allowedIdentities.emails**
      array of strings

      Exact email addresses this provider may assert.

      Compared case-insensitively.

      - **Element of the array**
        string
    - **spec.allowedIdentities.groups**
      array of strings

      Group names this provider may assert, with the `groupPrefix` in front when it is set.

      Applied on top of the provider-specific group filters (`allowedGroups`, `groups`, `teams`, `orgs[].teams`), if any.

      - **Element of the array**
        string

        Length: `1..256`

  - **spec.bitbucketCloud**
    object

    Parameters of the Bitbucket Cloud (intended for the `type: BitbucketCloud`).

    - **spec.bitbucketCloud.clientID**
      string

      Required value

      Team application ID from BitbucketCloud (Key).

    - **spec.bitbucketCloud.clientSecret**
      string

      Required value

Team application secret key from BitbucketCloud.

If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.bitbucketCloud.includeTeamGroups**
  boolean

  Optional parameter to include team groups.

  If enabled, the groups claim of Dex id_token will look like this:

  ```
  ["my_team", "my_team/administrators", "my_team/members"]
  ```

  Default: `false`

- **spec.bitbucketCloud.teams**
  array of strings

  A list of allowed Bitbucket Cloud teams (filter).

  The user token will contain a set intersection of Bitbucket Cloud teams and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain the user teams in the `groups` claim (similar to other providers).

- **spec.crowd**
  object

  Parameters of the Crowd (intended for the `type: Crowd`).

  - **spec.crowd.baseURL**
    string

    Required value

    Base part of Atlassian Crowd URL.

    Example:

    ```
    baseURL: https://crowd.example.com/crowd
    ```

  - **spec.crowd.clientID**
    string

    Required value

    Application ID from Atlassian Crowd (Application Name).

  - **spec.crowd.clientSecret**
    string

    Required value

    Application secret key from Atlassian Crowd (Password).

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.crowd.enableBasicAuth**
    boolean

    Enables Basic Authentication to the Kubernetes API server with the Crowd user's username and password.

    Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

    The parameter cannot be turned on while `twoFactor.providerTypes` includes `Crowd`, because Basic Authentication has no page for the second factor.

    Basic Authentication bypasses `spec.allowedIdentities`. Therefore only a `SuperAdmin` can create a provider with this parameter enabled or enable the parameter in an existing provider.

A successful check is cached together with the user's groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.crowd.groups**
  array of strings

  A list of allowed Crowd groups (filter).

  The user token will contain a set intersection of Crowd groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain all Crowd groups if the parameter is not set.

- **spec.crowd.usernamePrompt**
  string

  Prompt for username field.

  Default: `Crowd username`

- **spec.displayName**
  string

  Required value

  The provider name to show on the authentication provider selection page. The selection page will not be displayed if there is only one provider configured.

- **spec.enabled**
  boolean

  Whether the provider is enabled.

  A disabled provider is not shown on the sign-in page, and users cannot sign in through it. The DexProvider and its settings are kept.

  Default: `true`

- **spec.github**
  object

  Parameters of the GitHub provider (intended for the `type: Github` case only).

  - **spec.github.clientID**
    string

    Required value

    Organization application ID from GitHub.

  - **spec.github.clientSecret**
    string

    Required value

    Organization application secret key from GitHub.

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.github.orgs**
    array of objects

    Filter for user organizations. ID token will contain only organizations from this list. If the user is not in any organization from this list, an authorization will fail.

    By default, all organizations allowed.

    - **spec.github.orgs.name**
      string

      Required value

      Name of organization.

    - **spec.github.orgs.teams**
      array of strings

      A list of allowed GitHub teams (filter).

The user token will contain a set intersection of teams from GitHub and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

The user token will contain all GitHub teams if the parameter is not set.

- **spec.github.teamNameField**
  string

  As an example, group claims for member of ‘Site Reliability Engineers’ in Acme organization would yield:

  - [‘acme:Site Reliability Engineers’] for ‘name’
  - [‘acme:site-reliability-engineers’] for ‘slug’
  - [‘acme:Site Reliability Engineers’, ‘acme:site-reliability-engineers’] for ‘both’

  Default: `name`

  Allowed values: `name`, `slug`, `both`

- **spec.github.useLoginAsID**
  boolean

  Flag which will switch from using the internal GitHub id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.gitlab**
  object

  Parameters of the GitLab provider (intended for the `type: Gitlab` case only).

  - **spec.gitlab.baseURL**
    string

    Base part of GitLab URL.

    Example:

    ```
    baseURL: https://gitlab.example.com
    ```

  - **spec.gitlab.clientID**
    string

    Required value

    Application ID from GitLab.

  - **spec.gitlab.clientSecret**
    string

    Required value

    Application secret key from GitLab.

    If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.gitlab.groups**
    array of strings

    A list (filter) of allowed GitLab groups (**group paths** and not names).

    The user token will contain a set intersection of GitLab groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

    The user token will contain all GitLab groups if the parameter is not set;

  - **spec.gitlab.rootCAData**
    string

    A CA chain to validate the provider in PEM format.

    Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.gitlab.useLoginAsID**
  boolean

  Flag to switch from using the internal GitLab id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.groupPrefix**
  string

  Prefix that Dex puts in front of every group the provider asserts, whatever the type of the provider.

  With a prefix, the groups of this provider are named `<PREFIX><GROUP>` in the tokens and in the rules of the cluster, for example `corp-ldap:admins`. The prefix keeps them apart from the groups of another provider only when every provider that can emit such names has its own prefix.

  The prefix must end with a colon, as in `corp-ldap:`. It must differ from the prefix of every other DexProvider and of every TokenExchangeProvider group, and neither of two prefixes may start with the other. The admission webhooks refuse such a prefix.

  Every name in `allowedIdentities.groups` must start with the prefix, otherwise the API refuses the manifest, because Dex adds the prefix before it compares. The group settings of the provider type (for example, `oidc.allowedGroups`, `gitlab.groups` and `ldap.groupSearch`) take the names of the identity provider, without the prefix.

  The prefix cannot start with `system:`, which the Kubernetes API server reserves, or with `peering:`, which is reserved for the ServiceAccounts of the `d8-peering` namespace. It cannot be set together with `enableBasicAuth` of an `OIDC` or `Crowd` provider, because basic-auth-proxy checks the credentials with such a provider directly and passes its groups as they are.

  Setting or changing the prefix renames the groups of the provider, and the rules written for the old names stop applying to its users. Rewrite the rules before you set the prefix, as described in Prefix for the groups of a provider.

  By default, no prefix.

  Pattern: `^[A-Za-z0-9][A-Za-z0-9._:/-]*:$`

  Length: `2..64`

  Example:

  ```
  groupPrefix: 'corp-ldap:'
  ```

- **spec.ldap**
  object

  Parameters of the LDAP.

  - **spec.ldap.bindDN**
    string

    The DN for an application service account. The connector uses these credentials to search for users and groups. Not required if the LDAP server provides access for anonymous auth.

    Example:

    ```
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    ```

  - **spec.ldap.bindPW**
    string

    Password of the service account set in `bindDN`.

If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

Example:

```
bindPW: password
```

- **spec.ldap.enableBasicAuth**
  boolean

  Enables Basic Authentication to the Kubernetes API server with the LDAP user's username and password.

  Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

  The parameter cannot be turned on while `twoFactor.providerTypes` includes `LDAP`, because Basic Authentication has no page for the second factor.

  A successful check is cached together with the user's groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.ldap.groupSearch**
  object

  Group search queries for groups given a user entry. Details

  - **spec.ldap.groupSearch.baseDN**
    string

    Required value

    BaseDN to start the search from.

    Example:

    ```
    baseDN: cn=users,dc=example,dc=com
    ```

  - **spec.ldap.groupSearch.filter**
    string

    Optional filter to apply when searching the directory.

    Example:

    ```
    filter: "(objectClass=person)"
    ```

  - **spec.ldap.groupSearch.nameAttr**
    string

    Required value

    Represents group name.

    Example:

    ```
    nameAttr: name
    ```

  - **spec.ldap.groupSearch.userMatchers**
    array of objects

    Required value

    Following list contains field pairs that are used to match a user to a group. It adds a requirement to the filter that an attribute in the group must match the user's attribute value.

    - **spec.ldap.groupSearch.userMatchers.groupAttr**
      string

      Required value

      The name of the attribute that stores the group member names.

Example:

```
groupAttr: member
```

- **spec.ldap.groupSearch.userMatchers.recursionGroupAttr**
  string

  Name of the group attribute that stores the DNs of its members.

  When it is set, Dex also looks for the groups that contain the found groups, level by level, and adds them to the groups of the user. Use it for nested groups. The value is usually the same as `groupAttr`, for example `member`.

  Dex makes one more search for every group of every level at each sign-in and each refresh of the tokens, so keep `baseDN` and `filter` of `groupSearch` narrow.

  Example:

  ```
  recursionGroupAttr: member
  ```

- **spec.ldap.groupSearch.userMatchers.userAttr**
  string

  Required value

  The name of the attribute that stores the user name.

  Example:

  ```
  userAttr: uid
  ```

- **spec.ldap.host**
  string

  Required value

  Host and optional port of the LDAP server in the form “host:port”. If the port is not supplied, it will be guessed based on “insecureNoSSL”, and “startTLS” flags. 389 for insecure or StartTLS connections, 636 otherwise.

  Example:

  ```
  host: ldap.example.com:636
  ```

- **spec.ldap.insecureNoSSL**
  boolean

  Following field is required if the LDAP host is not using TLS (port 389). This option inherently leaks passwords to anyone on the same network as Dex.

  Mutually exclusive with `startTLS`, `insecureSkipVerify`, and `rootCAData`: when `insecureNoSSL` is true, TLS is disabled and those settings have no effect.

  Default: `false`

- **spec.ldap.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Has no effect when `insecureNoSSL` is true.

  Default: `false`

- **spec.ldap.kerberos**
  object

  Optional Kerberos (SPNEGO) SSO settings for LDAP.

  - **spec.ldap.kerberos.enabled**
    boolean

    Enable Kerberos SPNEGO flow for this LDAP provider.

Default: `false`

- **spec.ldap.kerberos.expectedRealm**
  string

  Optional realm to require (case-insensitive).

- **spec.ldap.kerberos.fallbackToPassword**
  boolean

  If `true`, in the absence of/incorrect `Authorization: Negotiate` header, a login/password form will be displayed instead of a 401 response with `WWW-Authenticate: Negotiate`.

  Default: `false`

- **spec.ldap.kerberos.keytabSecretName**
  string

  Name of the Secret in the `d8-user-authn` namespace containing the `krb5.keytab` key to mount for Dex.

  Required when `enabled` is `true`.

- **spec.ldap.kerberos.krb5ConfConfigMapName**
  string

  Not used. Dex validates Kerberos tickets with the keytab only, and the module does not mount this ConfigMap.

- **spec.ldap.kerberos.krb5ConfSecretName**
  string

  Not used. Dex validates Kerberos tickets with the keytab only, and the module does not mount this Secret.

- **spec.ldap.kerberos.usernameFromPrincipal**
  string

  How to map Kerberos principal to LDAP username.

  Default: `Localpart`

  Allowed values: `Localpart`, `SAMAccountName`, `UserPrincipalName`

- **spec.ldap.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Has no effect when `insecureNoSSL` is true.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.ldap.startTLS**
  boolean

  When connecting to the server, connect using the ldap:// protocol then issue a StartTLS command. If unspecified, connections will use the ldaps:// protocol.

  Mutually exclusive with `insecureNoSSL`.

  Default: `false`

- **spec.ldap.userSearch**
  object

  Required value

  User search maps a username and password entered by a user to a LDAP entry.
  Details...

  - **spec.ldap.userSearch.baseDN**

string

Required value

BaseDN to start the search from.

Example:

```
baseDN: cn=users,dc=example,dc=com
```

- **spec.ldap.userSearch.emailAttr**
  string

  Required value

  LDAP attribute that will be matched to Dex user email entry. When an email address is not available, use another value unique to the user, like uid.

  Example:

  ```
  emailAttr: mail
  ```

- **spec.ldap.userSearch.filter**
  string

  Optional filter to apply when searching the directory.

  Example:

  ```
  filter: "(objectClass=person)"
  ```

- **spec.ldap.userSearch.idAttr**
  string

  Required value

  LDAP attribute that will be matched to Dex user id entry.

  The binary GUID attributes of Active Directory, `objectGUID` and `mS-DS-ConsistencyGuid`, are supported: the user ID is then the GUID in its string form, for example `7ed41c2b-e283-4b10-9cb8-a6e7d1f39a4c`.

  Example:

  ```
  idAttr: uid
  ```

- **spec.ldap.userSearch.nameAttr**
  string

  LDAP attribute that will be matched to Dex user name entry. No default value provided.

  Example:

  ```
  nameAttr: name
  ```

- **spec.ldap.userSearch.username**
  string

  Username attribute used for comparing user entries. This will be translated and combined with the other filter as “(=)".

  Example:

  ```
  username: uid
  ```

- **spec.ldap.userSearch.usernames**
  array of strings

  Attributes that the entered username is compared with, instead of `username`.

  The search filter has the form `(|(<ATTR_1>=<USERNAME>)(<ATTR_2>=<USERNAME>))`, so a user can sign in, for example, with a login or an email. A sign-

in is refused if the username matches more than one entry.

The Kerberos principal mapped to a username (after `usernameFromPrincipal`) is compared with every attribute in the list the same way. List only attributes whose values are unique in the directory and consistent with `usernameFromPrincipal`. If a value of another entry equals the principal, SSO stops working for that user. A principal that matches none of the attributes of its own entry is matched to the entry of another user.

Set either `username` or `usernames`.

Example:

```
usernames:
- uid
- mail
```

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Length: `1..256`

- **spec.ldap.usernamePrompt**
  string

  The attribute to display in the provided password prompt. If unset, will display “LDAP Username”.

  Default: `LDAP username`

  Example:

  ```
  usernamePrompt: SSO Username
  ```

- **spec.oidc**
  object

  Parameters of the OIDC (intended for the `type: OIDC`).

  - **spec.oidc.allowedGroups**
    array of strings

    A list of groups the user must belong to in order to authenticate successfully.

    If none of the specified groups match the user’s groups, authentication will be denied.

    If the parameter is omitted, no group-based filtering will be applied.

  - **spec.oidc.basicAuthUnsupported**
    boolean

    Use POST requests to interact with the provider instead of including the token in the Basic Authorization header. Generally, Dex automatically determines the type of request to make, while in some cases enabling this parameter can help.

    Default: `false`

  - **spec.oidc.claimMapping**
    object

    Some providers return non-standard claims (eg. mail). Claim mappings are hints for Dex how to map claims to standard OIDC claims.

    Dex can only map a non-standard claim to a standard one if it’s not included in the id_token returned by OIDC provider.

    - **spec.oidc.claimMapping.email**
      string

      The claim to use as the user email.

      Default: `email`

    - **spec.oidc.claimMapping.groups**
      string

The claim to use as the user groups.

Default: `groups`

- **spec.oidc.claimMapping.preferred_username**
  string

  The claim to use as the user preferred username.

  Default: `preferred_username`

- **spec.oidc.claimMappingOverride**
  boolean

  If enabled, the claim mapping will override the standard OIDC claims.

  By default, the claim mapping will be used only if the standard OIDC claims are not present, e.g., if there is no `email` claim in the id_token, the `claimMapping.email` will be used.

  Default: `false`

- **spec.oidc.clientID**
  string

  Required value

  OIDC issuer application ID.

- **spec.oidc.clientSecret**
  string

  Required value

  OIDC issuer application secret key.

  If the caller is not allowed to read the `dexproviders/sensitive` subresource, the API returns `<omitted>` instead of the value.

- **spec.oidc.enableBasicAuth**
  boolean

  Enables Basic Authentication to the Kubernetes API server with the OIDC user’s username and password.

  Works only when the `apiserver.publishAPI` parameter of the `control-plane-manager` module is enabled. Only one DexProvider in the cluster can have `enableBasicAuth` enabled.

  Basic Authentication bypasses `spec.allowedIdentities`. Therefore only a `SuperAdmin` can create a provider with this parameter enabled or enable the parameter in an existing provider.

  A successful check is cached together with the user’s groups for 2 minutes, and a failed check for 10 seconds. A user whose access is revoked in the identity provider can use the Kubernetes API until the cached result expires.

- **spec.oidc.getUserInfo**
  boolean

  Request additional info about the authenticated user.

  See UserInfo endpoint.

  Default: `false`

- **spec.oidc.insecureSkipEmailVerified**
  boolean

  Allow authentication for clients without verified email address.

  Default: `false`

- **spec.oidc.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

Default: `false`

- **spec.oidc.issuer**
  string

  Required value

  Canonical URL of the provider, also used for configuration discovery. This value MUST match the value returned in the provider config discovery.

  Example:

  ```
  issuer: https://accounts.google.com
  ```

- **spec.oidc.promptType**
  string

  Determines if the Issuer should ask for confirmation and provide hints during the authentication process.

  By default, the confirmation will be requested on the first authentication. Possible values may vary depending on the Issuer.

  Default: `consent`

- **spec.oidc.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.oidc.scopes**
  array of strings

  List of additional scopes to request in token response.

  A scope is a single token and must not contain whitespace: the OAuth2 protocol passes scopes as a space-delimited list.

  Default: `["openid","profile","email","groups","offline_access"]`

  - **Element of the array**
    string

    Pattern: `^\S+$`

- **spec.oidc.upstreamLogout**
  boolean

  Signs the user out of the provider too when the user signs out of Dex.

  Works only with the browser sessions of Dex enabled and only when the provider advertises an `end_session_endpoint`.

  By default, signing out of Dex leaves the session of the provider alone: signing out of one application does not sign the user out of everything else that signs in through the provider. The provider may then sign the user in again without a password. To ask for the password at every sign-in, set `promptType: login`.

  Default: `false`

- **spec.oidc.userIDKey**
  string

  The claim to use as the user id.

  Default: `sub`

- **spec.oidc.userNameKey**
  string

The claim to use as the user name.

Default: `name`

- **spec.saml**
  object

  Parameters of a SAML provider (intended for the `type: SAML`).

  - **spec.saml.allowedGroups**
    array of strings

    If set, only members of at least one of the listed groups will be allowed to log in.

  - **spec.saml.emailAttr**
    string

    SAML attribute to map to the email claim.

    Default: `email`

  - **spec.saml.entityIssuer**
    string

    Issuer value that Dex sends in the SAML AuthnRequest (the entity ID of Dex as a service provider). Dex also expects this value in the audience of the SAML assertion.

    If not set, the request carries no Issuer, and Dex expects the callback URL in the audience.

  - **spec.saml.filterGroups**
    boolean

    If `true`, the connector will filter the groups claim to only include groups from the `allowedGroups` list.

    Default: `false`

  - **spec.saml.groupsAttr**
    string

    SAML attribute to map to the groups claim.

  - **spec.saml.groupsDelim**
    string

    Delimiter to split the groups attribute value if it is a single string.

  - **spec.saml.insecureSkipSignatureValidation**
    boolean

    Skip signature validation of SAML responses. This is insecure and should only be used for testing.

    Default: `false`

  - **spec.saml.nameIDPolicyFormat**
    string

    The NameIDPolicy format to request from the IdP.

    If not set, Dex requests `persistent`.

    A UserOperation `Lock` with `spec.target` applies to the user ID, which Dex takes from the NameID. Use `persistent` or `emailAddress` for such a lock to work. With `transient`, the IdP returns a new NameID at every sign-in, and a lock does not apply to the next sign-in of the user.

    Allowed values: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`

  - **spec.saml.redirectURI**
    string

    Callback URI.

    Overrides the default callback URL. Should be set to `https://<dex-domain>/callback`.

  - **spec.saml.rootCAData**

string

A CA chain to validate the provider in PEM format.

Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.saml.ssoIssuer**
  string

  Issuer value expected in the SAML response from the IdP.

  If not set, the Issuer of the response is not checked.

- **spec.saml.ssoURL**
  string

  Required value

  SSO URL used for POST value.

  Example:

  ```
  ssoURL: https://idp.example.com/saml/sso
  ```

- **spec.saml.usernameAttr**
  string

  SAML attribute to map to the username claim.

  Default: `name`

- **spec.tokenExchange**
  object

  Settings of token exchange (RFC 8693) through the provider. Dex exchanges an ID token of the provider's issuer for its own token. Only `OIDC` and `Gitlab` providers can exchange tokens.

  The provider's connector does not check the audience of the exchanged token, so any application of the issuer could get a cluster token. Exchange tokens through a TokenExchangeProvider instead. The field is temporary and will be removed after the next LTS release, together with token exchange through a DexProvider.

  - **spec.tokenExchange.enabled**
    boolean

    Whether the provider exchanges tokens. Absent means `false`.

    Exchange can only be turned off. An administrator cannot set `true`, either when creating the provider or when changing it. During the upgrade from a release without this field, `user-authn` sets it once on every existing `OIDC` and `Gitlab` provider. It sets `true` on all of them if at least one DexClient has the `dexclient.deckhouse.io/allow-access-to-kubernetes: "true"` annotation, and `false` on all of them otherwise. A DexAuthenticator with access to Kubernetes does not count. Its oauth2-proxy never exchanges tokens, and exchange with its client secret, taken from the Secret in the namespace of the application, is not kept.

- **spec.type**
  string

  Required value

  Type of authentication provider.

  Allowed values: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Group

Scope: Cluster
Version: v1alpha1

Contains information about a user group.

- **spec**
  object

  Required value

  - **spec.members**
    array of objects

    Required value

    List of group members.

    - **spec.members.kind**
      string

      Required value

      Type of a group member.

      Allowed values: `User`, `Group`

    - **spec.members.name**
      string

      Required value

      For kind User — User.metadata.name. For kind Group — the nested Group.spec.name (the name that appears in tokens), not metadata.name.

  - **spec.name**
    string

    Required value

    Unique group name that appears in the token `groups` claim and in RBAC subjects. Not `metadata.name`.

TokenExchangeProvider

Scope: Cluster
Version: v1alpha1

Defines a trusted issuer of tokens that Dex exchanges for its own tokens (RFC 8693 token exchange).

The name of the object is at most 50 characters.

A client named in `spec.clients` sends a token of the issuer to the Dex token endpoint and gets a Dex token for the cluster, which the Kubernetes API server accepts like the token of any other Dex user. Typical issuers are CI systems (GitLab CI `id_tokens`, GitHub Actions), the Dex of a neighbouring cluster, and the ServiceAccount token issuer of another cluster.

Dex checks the signature, the issuer, the expiry and the audience of the token and the required claims, then names the subject as `spec.identity` says. The provider serves token exchange only and is never shown on the sign-in page.

> **Warning.** Whoever may create the object chooses the issuer, its keys and the names, and so can get a token for any user or group outside the reserved prefixes. This right is equivalent to the right to impersonate users. Grant it to cluster administrators only.

Deleting the object makes the Kubernetes API refuse every token issued through it within seconds if the `enableMultiTenancy` parameter of the `user-authz` module is enabled and the cluster runs Kubernetes 1.34 or later. Otherwise these tokens stay valid until they expire (the `idTokenTTL` parameter of the module).

- **spec**
  object

  Required value

  - **spec.allowedIdentities**
    object

    Limits the identities this provider may assert, as in DexProvider. Dex refuses a subject whose

email is outside `emails` and `emailDomains` when either is set, and keeps only the groups listed in `groups` when it is set.

- **spec.allowedIdentities.emailDomains**
  array of strings

  Allowed email domains.

- **spec.allowedIdentities.emails**
  array of strings

  Allowed email addresses.

- **spec.allowedIdentities.groups**
  array of strings

  Allowed groups. The other groups are removed from the subject.

- **spec.audiences**
  array of objects

  Required value

  Audiences a token is accepted for. A token is accepted if its `aud` claim matches at least one entry.

  - **spec.audiences.exact**
    string

    Value that `aud` must equal.

  - **spec.audiences.prefix**
    string

    Prefix that `aud` must start with.

- **spec.claimValidation**
  array of objects

  Claim values a token must carry.

  A string claim must equal `equals`. A boolean or a number must equal `equals` when written as a string. A list must contain `equals`.

  - **spec.claimValidation.claim**
    string

    Required value

    Name of the claim to check.

  - **spec.claimValidation.equals**
    string

    Required value

    Value the claim must have.

- **spec.clients**
  array of objects

  Required value

  Dex clients that may exchange tokens through this provider. Other clients are refused before their token is checked.

  A listed DexClient can exchange tokens through this provider even when its `allowedProviders` does not name the provider.

  - **spec.clients.dexClient**
    object

    Required value

    DexClient that may exchange tokens. Its client ID is `dex-client-<DEXCLIENT_NAME>@<DEXCLIENT_NAMESPACE>`.

    - **spec.clients.dexClient.name**
      string

Required value

Name of the DexClient.

- **spec.clients.dexClient.namespace**
  string

  Required value

  Namespace of the DexClient.

- **spec.identity**
  object

  Required value

  How the subject of a token is named in the cluster. The Kubernetes API server takes the user name from the email, so the email is the user name unless `email` is set.

  - **spec.identity.email**
    object

    Email of the subject, made of `prefix` and the value of the string claim `claim`. The prefix is required and ends with a colon.

    - **spec.identity.email.claim**
      string

      Required value

      Name of the string claim whose value follows the prefix.

    - **spec.identity.email.prefix**
      string

      Prefix added to the value of the claim. It is required and ends with a colon, such as `ci:`.

  - **spec.identity.groups**
    array of objects

    Groups of the subject, taken from string or list claims.

    With `trimPrefix`, values that do not start with it are dropped, and it is removed from the others. Then `prefix` is added to every remaining value. It is required and ends with a colon.

    - **spec.identity.groups.claim**
      string

      Required value

      Name of the string or list claim that holds the groups.

    - **spec.identity.groups.prefix**
      string

      Prefix added to every group. It is required and ends with a colon, such as `ci:`.

    - **spec.identity.groups.trimPrefix**
      string

      Prefix that group values must start with. It is removed from them, and values without it are dropped.

  - **spec.identity.username**
    object

    Required value

    User name of the subject, made of `prefix` and the value of the string claim `claim`. The prefix is required and ends with a colon.

    - **spec.identity.username.claim**
      string

      Required value

      Name of the string claim whose value follows the prefix.

        - **spec.identity.username.prefix**
          string

          Prefix added to the value of the claim. It is required and ends with a colon, such as `ci:`.

  - **spec.issuer**
    string

    Required value

    Issuer of the tokens. It must equal the `iss` claim of a token.

    An https URL without a query or fragment.

    Example:

    ```
    issuer: https://gitlab.example.com
    ```

  - **spec.keys**
    object

    Required value

    Source of the keys that sign the tokens, either the discovery document of the issuer or public keys given here.

    - **spec.keys.discovery**
      object

      Settings for reading the keys from the discovery document of the issuer. Dex reads it at the first exchange and again after a failure.

      - **spec.keys.discovery.caData**
        string

        PEM bundle of the CA that the certificate of the issuer chains to, in addition to the system roots.

      - **spec.keys.discovery.url**
        string

        HTTPS address of the discovery document.

        If not set, `<ISSUER>/.well-known/openid-configuration` is used, where `<ISSUER>` is the value of `spec.issuer`.

    - **spec.keys.jwks**
      object

      Public keys of the issuer (JWK Set, RFC 7517). Use it when the issuer has no discovery document reachable from the cluster, or when several issuers share one URL, such as the ServiceAccount token issuers of different clusters. Private keys are refused. A token that names a key in its `kid` header is verified with the key of that `kid` only, even when the set has one key.

      - **spec.keys.jwks.keys**
        array of objects

        Required value

        Public keys in JWK format.

        - **spec.keys.jwks.keys.kty**
          string

          Required value

          Key type, such as `RSA` or `EC`.

  - **spec.subjectTokenMaxLifetime**
    string

    Longest time a token may be valid, counted from `iat` to `exp`, or from now to `exp` for a token without `iat`.

    A token valid for longer is refused, so that a leaked long-lived token cannot be exchanged for new Dex tokens until it expires. The value is set in hours, minutes and seconds, such as `24h`

or `1h30m`.

Default: `24h`

Length: `2..16`

- **status**
  object

  Status of the provider.

  While `metadata.deletionTimestamp` is set, the provider is being revoked, and a client must stop using the tokens it got through it.

  - **status.conditions**
    array of objects

    Conditions of the provider. The `Ready` condition shows whether the provider exchanges tokens, and its `reason` tells why it does not.

    - **status.conditions.lastTransitionTime**
      string

      Time the status last changed.

    - **status.conditions.message**
      string

      Details of the status.

    - **status.conditions.observedGeneration**
      integer

      Generation of the object that the condition was set for.

    - **status.conditions.reason**
      string

      Reason for the status, such as `PrefixConflict`.

    - **status.conditions.status**
      string

      Required value

      Status of the condition, `True`, `False` or `Unknown`.

    - **status.conditions.type**
      string

      Required value

      Type of the condition, such as `Ready`.

  - **status.connectorID**
    string

    ID of the Dex connector of the provider. Use it as `connector_id` in the exchange request. Tokens issued through it carry it in the `user-authn.deckhouse.io/dex-provider` extra of the user.

  - **status.observedGeneration**
    integer

    Generation of the object that the controller processed last.

UserAccount

Scope: Cluster
Version: v1alpha1

Controller-projected view of Dex Password, OfflineSessions, and UserIdentity objects for the Deckhouse web UI.

The controller keeps one UserAccount for each local user and one for each user who has signed in through any external provider.

Do not create or edit UserAccount objects. The `user-authn` controller owns them. The source of truth for login credentials and session state remains Dex.

- **spec**
  object

  Empty. UserAccount is controller-owned and has no user-editable specification.

  Do not create UserAccount objects. The controller projects it from Dex.

- **status**
  object

  Observed account state projected from Dex.

  This is a Deckhouse web UI view only and does not store credentials or connector secrets.

  - **status.connectorID**
    string

    Dex connector identifier.

    `local` for local accounts; otherwise the DexProvider name.

  - **status.email**
    string

    Account email as reported by Dex.

  - **status.expireAt**
    string

    Account expiration time, in RFC3339 format.

  - **status.groups**
    array of strings

    Groups associated with the account.

  - **status.incorrectLoginAttempts**
    integer

    Number of consecutive incorrect login attempts recorded in Dex.

    Dex counts them for local accounts, and for accounts of LDAP and Atlassian Crowd providers whose type is listed in the `passwordPolicy.lockout.applyToConnectors` module parameter.

    Allowed values: `0 <= X`

  - **status.kind**
    string

    Account kind.

    Possible values:

    - `Local`: Dex Password (static user).
    - `External`: Federated identity from a connector.

    Allowed values: `Local`, `External`

  - **status.lastLogin**
    string

    Time of the last sign-in through Dex, in RFC3339 format.

    Dex records it after the user passes every sign-in step, including 2FA. Basic Authentication to the API server through an LDAP provider with `enableBasicAuth` updates it too, because it goes through Dex.

    For an account that has not signed in since the upgrade to Deckhouse 1.78, this is the time of its last sign-in that issued a refresh token.

  - **status.locked**
    boolean

    Whether the account is currently locked.

  - **status.lockedByAdministrator**
    boolean

    Whether the current lock was set by an administrator (as opposed to an automatic password-

policy lockout).

This does not make the lock permanent. `lockedUntil` still defines when it expires.

- **status.lockedUntil**
  string

  Time until which the account remains locked, in RFC3339 format.

  After this time the lock is lifted automatically, including an administrator lock set for a limited duration (for example, one hour via UserOperation). A permanent administrator lock is stored as a far-future timestamp.

- **status.providerType**
  string

  External identity provider type.

  Equals `spec.type` of the DexProvider, for example `LDAP`, `Crowd`, `OIDC`, `SAML`, or `Github`.

  Empty for local accounts.

- **status.sessions**
  array of objects

  Devices signed in to the account, the most recently active first.

  A device is one of the following:

  - a browser signed in to Dex while the `sessions` of Dex are enabled, together with the applications whose refresh tokens are bound to its session (the DexAuthenticator objects, and DexClient objects with `refreshTokenLifetime: Session`);
  - a refresh token not bound to a live session of Dex in a browser: kubeconfig, a device flow, an application of a DexClient, Basic Authentication to the API server, a token whose session has ended while the application keeps it, and, while the sessions of Dex are disabled, every sign-in to an application of a DexAuthenticator.

  The list holds at most 50 devices. To sign a device out, create a UserOperation of the `Logout` type with its `id` in `logout.session` (a browser) or `logout.grant` (a refresh token).

  - **status.sessions.clients**
    array of strings

    Dex clients (applications) signed in on the device.

  - **status.sessions.endsAt**
    string

    Time when the session of the device ends unless the device is used again, in RFC3339 format. The session ends at the earlier of two limits: the lifetime counted from the sign-in and the time without use counted from the last use.

    - For a `Browser` device, this is the end of its Dex session (the `sessions.absoluteLifetime` and `sessions.validIfNotUsedFor` parameters). After it, the browser asks for the password at the next sign-in; the applications it has signed in to stay signed in.
    - For a `Grant` device, this is the end of its refresh token (the `refreshTokenAbsoluteLifetime` and `refreshTokenValidIfNotUsedFor` parameters). After it, the device signs in again.

    While the device is in use, the time moves forward and is updated with a delay of up to 15 minutes, so the session never ends before the time shown. Not set if the session has no end.

  - **status.sessions.id**
    string

    Required value

    Identifier of the device: the identifier of the session of Dex (the `sid` claim of its tokens) for a browser, the identifier of the refresh token for other devices.

  - **status.sessions.ipAddress**
    string

    Client address of the last sign-in of the device, as the Ingress controller passed it to Dex. It locates the device only approximately: a proxy, NAT, or VPN changes it. Behind a balancer that hides the address of the client, it is the address of the balancer (for details,

see the Which client address does Dex see? section).

Maximum length: `64`

- **status.sessions.kind**
  string

  Required value

  Kind of the device.

  Possible values:

  - `Browser`: a browser signed in to Dex.
  - `Grant`: a refresh token not bound to a live session of Dex in a browser.

  Allowed values: `Browser`, `Grant`

- **status.sessions.knownSince**
  string

  Time Dex first saw the browser of the device complete a sign-in to the account, in RFC3339 format.

  Set when more than a minute earlier than `signedInAt`: the browser signed in before, signed out, and signed in again. Dex recognizes the browser by a long-lived cookie it sets at sign-in. Signing out of the other devices on the account, signing out everywhere, and the operations that end every session of the user make Dex forget the browsers, and so does ending this device from another device or by an administrator.

- **status.sessions.lastActivity**
  string

  Time of the last use of the device, in RFC3339 format, rounded down to the hour. A device that signed in less than an hour ago shows its sign-in time.

- **status.sessions.signedInAt**
  string

  Time the device signed in, in RFC3339 format.

- **status.sessions.userAgent**
  string

  User-Agent header the device sent at its last sign-in, cut to 512 bytes.

  The device sets the header itself. Treat it as text.

  Maximum length: `512`

- **status.sessionsTruncated**
  boolean

  Whether the account has more devices than `sessions` lists. The least recently active ones are not listed.

- **status.twoFactor**
  object

  State of the 2FA of the account in Dex, without secrets.

  The field shows what the settings ask the users of the type of provider of the account for, whether the account has it, and each method it has.

  Set after the first sign-in of the account through Dex. Absent when the second factor does not cover the type of provider of the account.

  - **status.twoFactor.confirmed**
    boolean

    Whether the account has what `methods` and `require` ask for, with the TOTP secret confirmed by a code.

    A WebAuthn credential is complete at the registration.

    Until the code, a sign-in shows the QR code again.

  - **status.twoFactor.enrolled**
    boolean

Whether the account has what `methods` and `require` ask for: a TOTP secret, at least one WebAuthn credential, both (`All`) or one of them (`Any`).

A secret or a key of a method that is not asked for does not count. Without `methods`, whether the account has either. Dex creates the TOTP secret when it shows the user the QR code at a sign-in.

- **status.twoFactor.methods**
  array of strings

  Second factors the settings ask the users of the type of provider of the account for (`twoFactor.methods` of the module, or its `providerTypeOverrides` entry for the type).

  Absent when 2FA is off or does not cover the type.

  - **Element of the array**
    string

    Allowed values: `TOTP`, `WebAuthn`

- **status.twoFactor.require**
  string

  How the two `methods` combine. With `Any`, the user passes one of them, with `All`, both. Absent with one method.

  Allowed values: `Any`, `All`

- **status.twoFactor.totp**
  object

  TOTP secret of the account. Absent when there is none.

  A UserOperation of the `Reset2FA` type removes it, unless `spec.reset2FA.methods` lists only `WebAuthn`.

  - **status.twoFactor.totp.confirmed**
    boolean

    Whether the user has entered a code for the secret.

- **status.twoFactor.webAuthn**
  object

  WebAuthn credentials (security keys) of the account. Absent when there is none.

  A UserOperation of the `Reset2FA` type removes them, unless `spec.reset2FA.methods` lists only `TOTP`.

  - **status.twoFactor.webAuthn.credentials**
    integer

    Number of registered security keys.

  - **status.twoFactor.webAuthn.maxCredentials**
    integer

    How many credentials the account may register (`twoFactor.webAuthn.maxCredentials` of the module).

- **status.userID**
  string

  Unique user identifier from Dex.

- **status.userRef**
  string

  Name of the related User resource.

  Set for local accounts.

- **status.username**
  string

  Account username as reported by Dex.

UserOperation

Scope: Cluster

- v1
- v1alpha1

Describes a single-use operation to be applied to a user (password reset, 2FA reset, lock/unlock, logout).

A UserOperation runs once, and changes to its spec after creation are ignored. The module processes the operation and writes the result to `status.phase` (`Succeeded` or `Failed`). Operations are automatically deleted after the `retention.userOperations` setting of the module (24 hours by default). To perform a new action, create a new UserOperation.

`Lock`, `ResetPassword`, `Reset2FA`, and `Logout` end the sessions of the user at once. The module raises the generation of the user's Dex account (a provider and a user ID), records the revocation in a Dex Revocation object, and deletes the user's Dex AuthSession (browser session) and RefreshToken objects. Dex refuses everything it issued to the account before, and dex-authenticator stops accepting the user's sessions within seconds. These operations keep the user's 2FA secret. Only `Reset2FA` removes it, and it also deletes the user's Dex OfflineSessions object. `Unlock` and `RequirePasswordChange` end no sessions. What an operation revoked is in `status.revocation`.

- **spec**
  object

  Required value

  - **spec.initiatorType**
    string

    Indicates who initiated the operation. Possible values:

    - `admin`: By an administrator (e.g. via `d8 iam user` CLI or a manual UserOperation).
    - `system`: Reserved. No component sets this value (Dex locks a user after failed sign-in attempts without creating a UserOperation).
    - `self`: By the user (sign-out on all devices or password change in the Deckhouse web interface).
  - **spec.lock**
    object

    Parameters for locking the user.

    - **spec.lock.for**
      string

      Required value

      Lock duration as a Go-style duration string (`30s`, `10m`, `1h`, `2h30m`, `7d`), or `permanent` for an indefinite lock released only by `Unlock`. `1d` equals `24h`.

      Pattern: `^(permanent|([0-9]+(\.[0-9]+)?(s|m|h|d))+)$`
  - **spec.logout**
    object

    Narrows a `Logout` down. Without it, `Logout` ends every session of the user.

    Set at most one of the fields. `session` and `exceptSession` apply to Dex browser sessions.

    - **spec.logout.exceptGrant**
      string

      ID of the refresh token to keep, for signing out of the other devices from a device without a browser session of Dex. Every browser session of the user ends, except the one the kept refresh token was issued in, and every other refresh token is revoked. A token issued while the operation runs can survive it. The operation fails if the user has no such refresh token.

      Pattern: `^[a-z2-7]{1,64}$`
    - **spec.logout.exceptSession**
      string

      ID of the browser session to keep. Every other browser session of the user ends, and every refresh token not issued in the kept session is revoked. A token issued while the operation runs can survive it. The operation fails if the user has no such session.

      Pattern: `^[a-z2-7]{1,64}$`
    - **spec.logout.grant**

string

ID of the refresh token to revoke: a device signed in without a browser session of Dex, such as a kubeconfig. It is the `id` of a device of the `Grant` kind in `status.sessions` of the UserAccount and the `d8_gid` claim of the tokens issued with the refresh token. The other devices keep their sessions. If the user has no such refresh token, for example it has expired, has been revoked or belongs to another user, the operation ends nothing and succeeds with the `the user has no such grant` message.

Pattern: `^[a-z2-7]{1,64}$`

- **spec.logout.session**
  string

  ID of the browser session to end (the `sid` claim of the tokens issued in it), together with the refresh tokens issued in it. If the user has no such session, for example it has expired or belongs to another user, the operation ends nothing and succeeds with the `the user has no such session` message.

  Pattern: `^[a-z2-7]{1,64}$`

- **spec.reset2FA**
  object

  Narrows a `Reset2FA` down. Without it, `Reset2FA` removes every second factor of the user.

  - **spec.reset2FA.methods**
    array of strings

    Second factors to remove: `TOTP`, the secret of the authenticator application, and `WebAuthn`, the security keys. The other method stays. The user registers a removed method again at the next sign-in that asks for it. The sessions of the user end whichever methods are listed.

    - **Element of the array**
      string

      Allowed values: `TOTP`, `WebAuthn`

- **spec.resetPassword**
  object

  Parameters for password reset.

  - **spec.resetPassword.newPasswordHash**
    string

    Bcrypt hash of the new password, without Base64 encoding.

    If the caller is not allowed to read the `useroperations/sensitive` subresource, the API returns `<omitted>` instead of the value.

  - **spec.resetPassword.requirePasswordChange**
    boolean

    Require the user to change the password at the next sign-in.

    With `false`, the new password stays valid until the user changes it or until the rotation interval of the password policy ends. A password that does not meet the complexity rules still has to change at the next sign-in.

    Default: `true`

- **spec.target**
  object

  Identifier of an account of an external authentication provider, by email.

  The `Lock`, `Unlock`, `Logout`, and `Reset2FA` operations against `target` apply to the Dex account (UserIdentity object) of the user whose email at the last sign-in was `email`. `connectorID` and `userID` narrow the search down to one provider and one user ID. The operation fails if no account has the email (the user has never signed in) or if more than one account matches; then set `connectorID`, and `userID` if the provider has several users with the email. The operation applies only to the account found: a lock does not lock a local user or a user of another provider with the same email. A local user is addressed by `spec.user` only.

  The lock applies to the user ID that the provider returns at sign-in. A SAML provider must return the same ID at every sign-in, with `nameIDPolicyFormat` set to `persistent` or

`emailAddress` or not set. With `transient`, every sign-in gets a new ID, and the lock does not apply to the next sign-in.

> **i** Only one of the fields can be specified at the same time: `user` or `target`.

- **spec.target.connectorID**
  string

  ID of the DexProvider (connector) through which the user logs in.

  Examples:

  ```
  connectorID: my-ldap
  ```

  ```
  connectorID: my-crowd
  ```

- **spec.target.email**
  string

  Required value

  Email of the user, as reported by the provider at the user's last sign-in.

  The comparison is case-insensitive.

  Example:

  ```
  email: jane.doe@example.org
  ```

- **spec.target.userID**
  string

  ID of the user that the provider returns at sign-in (`userID` of the Dex UserIdentity object). It tells apart accounts of a provider with the same email.

  Example:

  ```
  userID: cn=jane.doe,ou=people,dc=example,dc=org
  ```

- **spec.type**
  string

  Required value

  Type of the user operation. Possible values:

  - `ResetPassword`: Password reset. Applies to local users (`spec.user`) only.
  - `Reset2FA`: Two-factor authentication reset. Supports both local users (`spec.user`) and external users (`spec.target`). Of the external users, only the users of an LDAP or Atlassian Crowd provider can have a second factor. For a user of a provider of another type, the operation only ends the sessions.
  - `Lock`: User lock. Supports both local users (`spec.user`) and external users (`spec.target`).
  - `Unlock`: User unlock. Supports both local users (`spec.user`) and external users (`spec.target`).
  - `Logout`: Ends the sessions of the user. Without `logout`, ends every session in every application and revokes the refresh tokens of kubeconfig files. Supports both local users (`spec.user`) and external users (`spec.target`).
  - `RequirePasswordChange`: The user has to set a new password at the next sign-in. Ends no sessions. Applies to local users (`spec.user`) only.

  Allowed values: `ResetPassword`, `Reset2FA`, `Lock`, `Unlock`, `Logout`, `RequirePasswordChange`

- **spec.user**
  string

  Name (username) of the target local User resource.

  Used for operations on users that are managed by DP via the User CRD (the built-in local connector).

> **i** Only one of the fields can be specified at the same time: `user` or `target`.

Example:

```
user: jane.doe
```

- **status**
  object
  - **status.completedAt**
    string

    Time when the operation was completed, in RFC 3339 format (e.g. `2025-01-15T10:30:00Z`). Empty until the operation finishes processing.
  - **status.message**
    string

    Optional message about the result or failure of the operation.
  - **status.phase**
    string

    Current phase of the operation. Possible values:

    - `Succeeded`: Operation was completed successfully.
    - `Failed`: Operation failed.

    Allowed values: `Succeeded`, `Failed`
  - **status.revocation**
    object

    What the operation revoked. Set by `Lock`, `ResetPassword`, `Reset2FA`, and `Logout`.

    - **status.revocation.authSessionsDeleted**
      integer

      Number of Dex AuthSession objects (browser sessions) of the user the operation deleted.
    - **status.revocation.generation**
      integer

      New generation of the user's Dex account. Everything issued to the account at a lower generation is revoked. Not set when the operation did not revoke the whole account.
    - **status.revocation.name**
      string

      Name of the Dex Revocation object the operation wrote, or of the first of them.
    - **status.revocation.refreshTokensDeleted**
      integer

      Number of Dex RefreshToken objects of the user the operation deleted.

Deprecated resource. Support for the resource might be removed in a later release.

Describes an operation to be applied to a user (password reset, 2FA reset, lock/unlock).

The `Lock`, `ResetPassword`, and `Reset2FA` operations end the sessions of the user at once, as in `v1`. These operations keep the user's 2FA secret. Only `Reset2FA` removes it, and it also deletes the user's Dex OfflineSessions object.

- **spec**
  object

  Required value
  - **spec.initiatorType**
    string

    Indicates who initiated the operation. Possible values:

    - `admin`: By an administrator.
    - `system`: Reserved. No component sets this value.

    - `self`: By the user.
  - **spec.lock**
    object

    Parameters for locking the user.

    - **spec.lock.for**
      string

      Required value

      Lock duration as a Go-style duration string (`30s`, `10m`, `1h`, `2h30m`, `7d`), or `permanent` for an indefinite lock released only by `Unlock`. `1d` equals `24h`.

      Pattern: `^(permanent|([0-9]+(\.[0-9]+)?(s|m|h|d))+)$`
  - **spec.resetPassword**
    object

    Parameters for password reset.

    - **spec.resetPassword.newPasswordHash**
      string

      Bcrypt hash of the new password.

      If the caller is not allowed to read the `useroperations/sensitive` subresource, the API returns `<omitted>` instead of the value.
  - **spec.type**
    string

    Required value

    Type of the user operation. Possible values:

    - `ResetPassword`: Password reset.
    - `Reset2FA`: Two-factor authentication reset.
    - `Lock`: User lock.
    - `Unlock`: User unlock.

    Allowed values: `ResetPassword`, `Reset2FA`, `Lock`, `Unlock`
  - **spec.user**
    string

    Required value

    Name (username) of the target User resource.

    Example:

    ```
    user: jane.doe
    ```

- **status**
  object
  - **status.completedAt**
    string

    Time when the operation was completed.
  - **status.message**
    string

    Optional message about the result or failure of the operation.
  - **status.phase**
    string

    Current phase of the operation. Possible values:

    - `Succeeded`: Operation completed successfully.
    - `Failed`: Operation failed.

    Allowed values: `Succeeded`, `Failed`

User

Scope: Cluster

- v1
- v1alpha1

Contains information about the static user.

Creating a user

- **spec**
  object

  Required value

  - **spec.email**
    string

    Required value

    User email. Must be in lowercase and at most 150 bytes long (150 characters of Latin letters, digits and punctuation): Dex names the object that keeps the password of the user after the email, and the name of a Kubernetes object is at most 253 characters.

    To grant this user access, use the `email` value as the `User` subject name in a RoleBinding or ClusterRoleBinding of the granular role-based model.

    Important! Email addresses are case-insensitive for uniqueness validation. Dex automatically converts emails to lowercase during authentication. For new users, uppercase letters in email addresses are not allowed. Existing users with uppercase emails can be updated without changing the email field.

    Length: `1..150`

    Example:

    ```
    email: user@domain.com
    ```

  - **spec.groups**
    Deprecated
    array of strings

    Static user groups.

    Since the parameter has been deprecated, use the Group resource to add users to groups.

  - **spec.password**
    string

    Required value

    User password hash in plaintext or Base64 encoded.

    Use the following command to encode the password hash in Base64: `echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. You can also create the user with `d8 iam user create`, which computes the hash locally.

    > **Caution.** If the password does not meet the complexity rules of the password policy, the user has to change it at the first sign-in. To make the user change the password at the first sign-in in any case, set `requirePasswordChangeOnFirstLogin`.

    > **Caution.** This field is immutable after the User is created. Changing the password via `kubectl edit` does not update Dex credentials and will be rejected. To reset the password, use the UserOperation resource (or `d8 iam user`).

    If the caller is not allowed to read the `users/sensitive` subresource, the API returns `<omitted>` instead of the value.

    Example:

```
password:
JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6Szhp
eS5H
```

- **spec.requirePasswordChangeOnFirstLogin**
  boolean

  Require the user to change the password at the first sign-in.

  Applies when the module creates the Dex credentials of the user from `password`: when the User is created, created again, or restored from a backup.

  Can be set only when the User is created. Adding, changing, or removing the field later is rejected.

- **spec.ttl**
  string

  Static user TTL.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  You can only set the TTL once. The `expireAt` date will not be updated if you change it again.

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  Example:

  ```
  ttl: 24h
  ```

- **spec.userID**
  Deprecated
  string

  Deprecated. Ignored if set. Dex Password.userID is User.metadata.name.

- **status**
  object
  - **status.expireAt**
    string

    User account expiration time. Set only when `spec.ttl` is set. At this time the User is deleted and removed from all Groups.

  - **status.groups**
    array of strings

    Groups the user belongs to, computed from Group membership. Values are Group.spec.name.

  - **status.lock**
    object

    Information about user account lock. If present, the user cannot authenticate until the specified time or until the lock is manually lifted by an administrator.

    Actions on the account (lock, unlock, password reset, forced password change, 2FA reset, logout) are performed via the UserOperation resource. A lock, a password reset, a 2FA reset and a logout end the sessions of the user at once. After a password reset or a forced password change, the user is presented with a password change form on the next successful login. Only a 2FA reset removes the user’s 2FA secret.

    - **status.lock.message**
      string

      Human-readable explanation of the lock reason.

    - **status.lock.reason**
      string

      Machine-readable reason for the account lock. Used for programmatic checks and policy enforcement.

      Allowed values: `PasswordPolicyLockout`, `LockedByAdministrator`

    - **status.lock.state**
      boolean

Indicates whether the user account is currently locked.

- `true` — account is locked.
- `false` — account is active.

- **status.lock.until**
  string

  The time until which the user is locked, in RFC3339 format. After this time, the account lock will be automatically lifted.

Deprecated resource. Support for the resource might be removed in a later release.

Contains information about the static user.

Creating a user

- **spec**
  object

  Required value

  - **spec.email**
    string

    Required value

    User email. Must be in lowercase and at most 150 bytes long (150 characters of Latin letters, digits and punctuation): Dex names the object that keeps the password of the user after the email, and the name of a Kubernetes object is at most 253 characters.

    To grant this user access, use the `email` value as the `User` subject name in a RoleBinding or ClusterRoleBinding of the granular role-based model.

    Important! Email addresses are case-insensitive for uniqueness validation. Dex automatically converts emails to lowercase during authentication. For new users, uppercase letters in email addresses are not allowed. Existing users with uppercase emails can be updated without changing the email field.

    Length: `1..150`

    Example:

    ```
    email: user@domain.com
    ```

  - **spec.groups**
    Deprecated
    array of strings

    Static user groups.

    Since the parameter has been deprecated, use the Group resource to add users to groups.

  - **spec.password**
    string

    Required value

    User password hash in plaintext or Base64 encoded.

    Use the following command to encode the password hash in Base64: `echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. You can also create the user with `d8 iam user create`, which computes the hash locally.

    > **Caution.** If the password does not meet the complexity rules of the password policy, the user has to change it at the first sign-in. To make the user change the password at the first sign-in in any case, set `requirePasswordChangeOnFirstLogin`.

    > **Caution.** This field is immutable after the User is created. Changing the password via `kubectl edit` does not update Dex credentials and will be rejected. To reset the password, use the UserOperation resource (or `d8 iam user`).

    If the caller is not allowed to read the `users/sensitive` subresource, the API returns `<omitted>` instead of the value.

Example:

```
password:
JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6Szhp
eS5H
```

- **spec.requirePasswordChangeOnFirstLogin**
  boolean

  Require the user to change the password at the first sign-in.

  Applies when the module creates the Dex credentials of the user from `password`: when the User is created, created again, or restored from a backup.

  Can be set only when the User is created. Adding, changing, or removing the field later is rejected.

- **spec.ttl**
  string

  Static user TTL.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  You can only set the TTL once. The `expireAt` date will not be updated if you change it again.

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  Example:

  ```
  ttl: 24h
  ```

- **spec.userID**
  string

  Deprecated. Ignored if set. Dex Password.userID is User.metadata.name.

  Example:

  ```
  userID: '08a8684b-db88-4b73-90a9-3cd1661f5466'
  ```

- **status**
  object
  - **status.expireAt**
    string

    User account expiration time. Set only when `spec.ttl` is set. At this time the User is deleted and removed from all Groups.

  - **status.groups**
    array of strings

    Groups the user belongs to, computed from Group membership. Values are Group.spec.name.

  - **status.lock**
    object

    Information about user account lock. If present with `state: true`, the user cannot authenticate until the specified time or until the lock is manually lifted by an administrator.

    Actions on the account (lock, unlock, password reset, forced password change, 2FA reset, logout) are performed via the UserOperation resource. A lock, a password reset, a 2FA reset and a logout end the sessions of the user at once. After a password reset or a forced password change, the user is presented with a password change form on the next successful login. Only a 2FA reset removes the user’s 2FA secret.

    - **status.lock.message**
      string

      Human-readable explanation of the lock reason.

    - **status.lock.reason**
      string

      Machine-readable reason for the account lock. Used for programmatic checks and policy

enforcement.

Allowed values: `PasswordPolicyLockout`, `LockedByAdministrator`

- **status.lock.state**
  boolean

  Indicates whether the user account is currently locked.

  - `true` — account is locked.
  - `false` — account is active.
- **status.lock.until**
  string

  The time until which the user is locked, in RFC3339 format. After this time, the account lock will be automatically lifted.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The user-authn module: usage

> The module lifecycle stage: General Availability

An example of the module configuration

The example adds a direct access method to the API server for the kubeconfig that users generate in the Deckhouse web UI.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    kubeconfigGenerator:
    - id: direct
      masterURI: https://192.0.2.10:6443
      description: "Direct access to kubernetes API"
```

Configuring a provider

In the examples below, `<DEX_DOMAIN>` is the Dex domain. It is the `publicDomainTemplate` value with `%s` replaced by `dex`, for example `dex.example.com`.

Checking provider connectivity

The provider page in the Deckhouse web UI has a “Check connection” action. It creates a DexProviderCheck resource and waits for the `user-authn` controller to write the result to its `status`.

The check verifies that:

- the referenced DexProvider exists and is enabled;
- the Dex discovery endpoint inside the cluster is reachable;
- the external provider endpoint is reachable.

For OIDC providers, the check also reads the discovery document and JWKS. For LDAP providers, it verifies TCP/TLS/StartTLS reachability and, when Kerberos is enabled, that the keytab Secret exists. The check also verifies that the provider accepts the client or service account credentials (GitHub, GitLab,

Bitbucket Cloud, Crowd, OIDC, LDAP bind) and warns about disabled TLS verification and certificates that expire soon. The check does not perform a full login flow and does not validate a test user password.

To run the check without the web UI, create a DexProviderCheck:

```
apiVersion: deckhouse.io/v1
kind: DexProviderCheck
metadata:
  name: check-github
spec:
  providerName: github
```

Then read its status:

```
d8 k get dexproviderchecks check-github -o yaml
```

Limiting the identities of a provider

A user who signs in through a DexProvider gets the email and the groups that the identity provider reports. The email is the Kubernetes username. So the provider gives its users every permission already granted in the cluster to these emails and groups.

To limit them, use `spec.allowedIdentities`. The `emails` and `emailDomains` lists limit the email, and Dex refuses sign-in for any other email. The `groups` list limits the groups. Dex keeps only the listed groups in the token and refuses sign-in if none is left.

The limits apply to every provider type, in addition to the group filters of the type (`oidc.allowedGroups`, `gitlab.groups`, `crowd.groups`, `bitbucketCloud.teams` without `includeTeamGroups`, `github.orgs[].teams` in every organization, `saml.allowedGroups` with `filterGroups: true`). Group names are compared exactly. Nesting of Group objects does not extend the list, because a token from an external provider carries only the groups that the provider reports.

Dex checks the limits at sign-in, at every refresh-token request and at Basic Authentication to the API server through an LDAP provider. Dex revokes a refresh token that falls outside the limits. Narrowing the limits therefore ends the existing sessions of the users it excludes, and they have to sign in again. For a Crowd or OIDC provider, basic-auth-proxy checks the credentials with the provider directly, so the limits do not apply to Basic Authentication.

The same limits define who can create or change the provider. You must be able to assign every role already granted to the emails and groups that the provider can report. A provider without an email limit can report the email of a SuperAdmin, so only a SuperAdmin can create or change it. A Crowd or OIDC provider with `enableBasicAuth: true` counts as a provider without an email limit, because Basic Authentication through it bypasses the limits. Only a SuperAdmin can create such a provider or turn on `enableBasicAuth` in an existing one.

Example of a provider limited by an email domain and a group:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: contractors
spec:
  type: OIDC
  displayName: Contractors
  oidc:
    issuer: https://idp.contractor.example
    clientID: dex
    clientSecret: secret
  allowedIdentities:
    emailDomains: [contractor.example]
    groups: [contractors]
```

The following changes are not checked: rotating `clientSecret` or `bindPW`, changing `displayName`, disabling the provider, narrowing the limits, applying the same manifest again, and deleting the provider. Any other change is checked as a new provider, including enabling a disabled provider.

If the check fails, the error lists the roles that the provider could reach, for example:

```
dexproviders.deckhouse.io "corp": the provider can assert identities that already carry roles [user-
authz:super-admin]; the requester's can-assign range is basic<=ClusterAdmin and does not cover them.
Narrow spec.allowedIdentities (emails, emailDomains, groups) or ask a SuperAdmin
```

Prefix for the groups of a provider

The cluster knows the groups a provider reports by their names, so a rule for the group `admins` applies to every member of any group `admins`. When two providers report a group of the same name, for example an LDAP directory and a GitLab, the members of both get the rights of the rule.

`spec.groupPrefix` of a DexProvider puts a prefix in front of every group the provider asserts, whatever its type. With `groupPrefix: corp-ldap:`, the group `admins` of the directory is `corp-ldap:admins` in the tokens, in the rules (the subjects of the AuthorizationRule objects) and in the `allowedGroups` of DexClient and DexAuthenticator. A prefix keeps the groups of the providers apart only when every provider that can emit such names has its own prefix. A provider without a prefix passes the group names as they are (OIDC, a GitHub team named `org:team`, GitLab, SAML and the local groups) and can emit a name that starts with the prefix. When the rules rely on prefixed groups, set a prefix on every provider.

The following provider prefixes the groups of an LDAP directory with `corp-ldap:`:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: corp-ldap
spec:
  type: LDAP
  displayName: Corp
  groupPrefix: "corp-ldap:"
  allowedIdentities:
    groups:
    - "corp-ldap:admins"
    - "corp-ldap:developers"
  ldap:
    host: ldap.example.com:636
    userSearch:
      baseDN: ou=people,dc=example,dc=com
      username: mail
      idAttr: DN
      emailAttr: mail
```

Keep the following in mind when you set a prefix:

- There is no prefix by default. The prefix renames the groups, and the rules written for the old names stop applying to the users of the provider and so would lock them out. An existing provider takes the prefix only when you set it. Rewrite the rules first, then set the prefix.
- Rewrite the subjects of ClusterAuthorizationRule and AuthorizationRule, the groups of ClusterRoleBinding and RoleBinding, `auth.allowedUserGroups` in the ModuleConfig of `openvpn`, `documentation`, `istio`, `cilium-hubble` and `deckhouse-tools` (those modules render a DexAuthenticator from the setting), `allowedIdentities.groups` of the DexProvider and `allowedGroups` of DexAuthenticator and DexClient.
- After the prefix is set or changed, the users of the provider sign in again, and the new tokens carry the prefix.
- The access and ID tokens already issued keep the old names only until they expire. A rule that still names a group without the prefix, including a cluster-admin rule, stops matching as soon as those tokens are gone.
- A token that a Dex pod still running the old configuration issues later than 30 minutes after the change can keep the old names. The rules rewritten for the prefix do not match them.
- The lists that name the groups of Dex take the names with the prefix: `allowedIdentities.groups` of the provider, and `allowedGroups` of a DexClient or a DexAuthenticator. Every name in `allowedIdentities.groups` must start with `groupPrefix`, otherwise the API refuses the manifest, because Dex adds the prefix before it compares.
- The group settings of the provider type (for example, `oidc.allowedGroups`, `gitlab.groups`, `crowd.groups` and `ldap.groupSearch`) are about the groups of the identity provider and take its names, without the prefix.
- A group that already starts with the prefix keeps its name.
- The prefix ends with a colon, as in `corp-ldap:`, and differs from the prefix of every other DexProvider and from the group prefixes of every TokenExchangeProvider. Neither of two prefixes

may start with the other. The admission webhooks refuse a prefix that breaks this, whichever of the objects is created or changed last.
- The prefix cannot start with `system:`, which the Kubernetes API server reserves, or with `peering:`, which is reserved for the ServiceAccounts of the `d8-peering` namespace. It cannot be set together with `enableBasicAuth` of an `OIDC` or `Crowd` provider, because basic-auth-proxy checks the credentials with such a provider directly and passes its groups as they are, so the prefix would not apply to them. Basic Authentication of an `LDAP` provider goes through Dex and gets the prefix.
- With a prefix, the provider can assert only groups that start with it. Therefore the check of who may create or change the provider (“Limiting the identities of a provider”) counts, of the roles granted to groups, only the roles of the groups with this prefix.

GitHub

The example shows the provider’s settings for integration with GitHub.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company Github
  # Optional: disable the provider without deleting CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

In your GitHub organization, register an OAuth application. To do this, go to `Settings` → `Developer settings` → `OAuth Apps` → `New OAuth App` and set `Authorization callback URL` to `https://<DEX_DOMAIN>/callback`.

Paste the generated `Client ID` and `Client Secret` into the DexProvider.

If the organization restricts third-party applications, request approval in `Settings` → `Applications` → `Authorized OAuth Apps` → `<APPLICATION_NAME>` → `Send Request`, and ask an organization owner to approve the request.

GitLab

The example shows the provider’s settings for integration with GitLab.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated GitLab
  gitlab:
    baseURL: https://gitlab.example.com
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Create a new application in GitLab.

To do this:

- **Self-hosted**: Go to `Admin area` → `Application` → `New application`, set `Redirect URI (Callback URL)` to `https://<DEX_DOMAIN>/callback` and select the `read_user` and `openid` scopes.
- **Cloud GitLab.com**: Under the main project account, go to `User Settings` → `Application` → `New`

`application`, set `Redirect URI (Callback URL)` to `https://<DEX_DOMAIN>/callback` and select the `read_user` and `openid` scopes.
- **GitLab 16 and later**: Enable the `Trusted`/`Trusted applications are automatically authorized on GitLab OAuth flow` checkbox when creating the application.

Paste the generated `Application ID` and `Secret` into the DexProvider.

Atlassian Crowd

The example shows the provider's settings for integration with Atlassian Crowd.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseURL: https://crowd.example.com/crowd
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Create a new `Generic` application in the corresponding Atlassian Crowd project. To do this, go to `Applications` → `Add application`.

Paste the generated `Application Name` and `Password` into the DexProvider.

Specify Crowd groups in the DexProvider in lowercase.

Bitbucket Cloud

The example shows the provider's settings for integration with Bitbucket Cloud.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
    teams:
    - administrators
    - users
```

Create a new OAuth consumer in the Bitbucket team menu. To do this, go to `Settings` → `OAuth consumers` → `New application`, set `Callback URL` to `https://<DEX_DOMAIN>/callback` and allow access for `Account: Read` and `Workspace membership: Read`.

Paste the generated `Key` and `Secret` into the DexProvider.

OIDC (OpenID Connect)

Authentication through an OIDC provider requires registering a client (or "creating an application"). Register it as the provider's documentation describes (for example, Okta, Keycloak, Gluu).

Paste the generated `clientID` and `clientSecret` into the DexProvider.

Keycloak

After selecting a `realm` to configure, adding a user in the Users section, and creating a client in the Clients section with authentication enabled, which is required to generate the `clientSecret`, you need to perform the following steps:

- Create a `scope` named `groups` in the Client scopes section and assign it the predefined mapping `groups`. (“Client scopes” → “Client scope details” → “Mappers” → “Add predefined mappers”)
- In the previously created client, add this `scope` in the Client scopes tab (“Clients” → “Client details” → “Client Scopes” → “Add client scope”).
- In the “Valid redirect URIs”, “Valid post logout redirect URIs”, and “Web origins” fields of the client configuration, specify `https://<DEX_DOMAIN>/*`.

The example shows the provider’s settings for integration with Keycloak.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Use the name of your
realm
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups
```

If email verification is not enabled in Keycloak, to properly use it as an identity provider, adjust the `Client Scopes` settings in one of the following ways:

- Delete the `Email verified` mapping (“Client Scopes” → “Email” → “Mappers”). This is required for proper processing of the `insecureSkipEmailVerified` field when it’s set to `true` and for correct permission assignment to users with unverified emails.
- If you can’t modify or delete the `Email verified` mapping, create a new Client Scope named `email_dkp` (or any other name) and add two mappings:
  - `email`: “Client Scopes” → `email_dkp` → “Add mapper” → “From predefined mappers” → `email`.
  - `email verified`: “Client Scopes” → `email_dkp` → “Add mapper” → “By configuration” → “Hardcoded claim”. Specify the following fields:
    - “Name”: `email verified`
    - “Token Claim Name”: `emailVerified`
    - “Claim value”: `true`
    - “Claim JSON Type”: `boolean`

  After that, in the client registered for the Deckhouse Platform cluster in “Clients”, change `Client scopes` from `email` to `email_dkp`.

  In the DexProvider resource, specify `insecureSkipEmailVerified: true` and in the `.spec.oidc.scopes` field, change the Client Scope name to `email_dkp` following the example:

  ```
    scopes:
      - openid
      - profile
      - email_dkp
      - groups
  ```

⚠ With `insecureSkipEmailVerified: true`, Dex accepts unverified email addresses. Permissions are granted by email, so use this parameter only if the identity provider does not let users set an arbitrary email address.

Okta

The example shows the provider's settings for integration with Okta.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientID: plainstring
    clientSecret: plainstring
    getUserInfo: true
```

Blitz Identity Provider

When you register the application in the Blitz Identity Provider, specify `https://<DEX_DOMAIN>/callback` as the URL to redirect the user to after authorization.

The example below shows the provider settings for integration with Blitz Identity Provider.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim for getting user groups, configured on the Blitz
    clientID: clientID
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
    - profile
    - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC
```

For the application logout to work correctly (the token is revoked and the user must sign in again), set the `promptType` parameter to `login`.

To ensure granular user access to applications, you have to:

- Add the `allowedUserGroups` parameter to the `ModuleConfig` of the target application.
- Add the user to the groups (group names should be the same for Blitz and Deckhouse).

Below is an example for prometheus:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access
```

LDAP

The example shows the provider’s settings for integration with Active Directory.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----

    bindDN: cn=dex-readonly,cn=users,dc=example,dc=com
    bindPW: <PASSWORD>

    usernamePrompt: Email Address

    enableBasicAuth: true

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
      - userAttr: DN
        groupAttr: member
      nameAttr: cn
```

To configure authentication, create a read-only service account in LDAP. Specify its DN and password in the `bindDN` and `bindPW` fields of the DexProvider. Enter the password in `bindPW` in plain text. Hashed passwords are not supported.

If LDAP allows anonymous read access, you can omit `bindDN` and `bindPW`.

Nested groups

By default, Dex adds to the token only the groups of which the user is a direct member. A group that contains the user's group is not added. To add the groups of the higher levels, set `recursionGroupAttr` in `userMatchers` to the attribute of a group that stores its members. Dex then looks for the groups that contain each found group, level by level, and adds them to the groups of the user. A cycle of groups ends the search.

The following `groupSearch` adds the groups of every level through the `member` attribute:

```
groupSearch:
  baseDN: cn=Users,dc=example,dc=com
  filter: "(objectClass=group)"
  userMatchers:
  - userAttr: DN
    groupAttr: member
    recursionGroupAttr: member
  nameAttr: cn
```

Dex makes one more search for every group of every level at each sign-in and each refresh of the tokens. Keep `baseDN` and `filter` of `groupSearch` narrow, and expect a longer sign-in on a deep hierarchy.

Signing in with one of several attributes

`userSearch.username` names one attribute that the entered username is compared with. To let users sign in with a login or an email, list the attributes in `userSearch.usernames` instead of `username` (set one of the two parameters). Dex searches for an entry in which any of them matches. A sign-in is refused if the username matches more than one entry.

The following `userSearch` lets users sign in with `sAMAccountName` or `userPrincipalName`:

```
userSearch:
  baseDN: cn=Users,dc=example,dc=com
  filter: "(objectClass=person)"
  usernames: [sAMAccountName, userPrincipalName]
  idAttr: DN
  emailAttr: userPrincipalName
  nameAttr: cn
```

Configuring Basic Authentication

To enable Basic Authentication for the Kubernetes API using LDAP credentials:

1. Ensure that the `apiserver.publishAPI` parameter is enabled in the `control-plane-manager` module configuration.
2. Set `enableBasicAuth: true` in the LDAP DexProvider.

> **i** Only one provider in the cluster can have `enableBasicAuth` enabled.

After configuration, users can access the Kubernetes API via `kubectl`, using their LDAP username and password.

Example kubeconfig for the user:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Path to CA certificate or insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default
```

Kerberos (SPNEGO) SSO for LDAP

Dex supports passwordless Kerberos (SPNEGO) flow for the LDAP connector. When enabled, a browser that trusts the Dex host will send `Authorization: Negotiate …` and Dex will validate the Kerberos ticket using a service keytab, skip the login form, map the Kerberos principal to an LDAP username, query groups, and complete the OIDC flow.

The principal mapped to a username is compared with every attribute listed in `userSearch.usernames` (or with `username`, when that is set), with the same filter as a password sign-in. With `usernameFromPrincipal: UserPrincipalName`, the principal is compared as `user@REALM`, with `Localpart` or `SAMAccountName` as the part before `@`. The value must match an attribute of the user's own entry. List only attributes whose values are unique in the directory and consistent with `usernameFromPrincipal`. If a value of another entry equals the principal, SSO stops working for that user (`multiple results`). A principal that matches none of the attributes of its own entry is matched to the entry of another user. The password is not checked on this path.

Minimal example extending the LDAP provider spec:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=dex-readonly,cn=users,dc=example,dc=com
    bindPW: <PASSWORD>
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
      - userAttr: uid
        groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab  # Secret in d8-user-authn with key
'krb5.keytab'
      expectedRealm: EXAMPLE.COM           # optional, case-insensitive match
      usernameFromPrincipal: SAMAccountName  #
Localpart|SAMAccountName|UserPrincipalName
      fallbackToPassword: false            # default false; if true, render form when
header missing/invalid
```

Notes:

- The Secret `dex-kerberos-keytab` must exist in the `d8-user-authn` namespace and have a data key named exactly `krb5.keytab`.
- A single Dex Pod can serve multiple LDAP+Kerberos providers. Each provider mounts its own keytab; a shared `krb5.conf` is not required (Dex validates tickets offline using the keytab).
- A DexProvider with `enabled: true` must have `keytabSecretName` set, otherwise the object is rejected when saved. To turn Kerberos off and keep its settings, set `enabled: false`.

SAML

The example shows the provider’s settings for integration with a SAML 2.0 Identity Provider (e.g., AD FS, Okta, Keycloak).

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Corporate SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent
```

To configure the SAML Identity Provider:

1. Register Dex as a Service Provider (SP) with your identity provider using the following settings:
   - **ACS URL (Assertion Consumer Service)**: `https://<DEX_DOMAIN>/callback`
   - **Entity ID**: `https://<DEX_DOMAIN>/callback`
   - **Name identifier format**: `persistent` or `emailAddress`
2. Configure attribute mapping in the identity provider to send `email`, `name` (username), and `groups` attributes in the SAML assertion.
3. Export the identity provider signing certificate and specify it in the `rootCAData` field of the DexProvider resource.

> **i** SAML does not natively support refresh tokens. Dex caches the user identity from the initial SAML assertion and returns it on subsequent refresh requests. The session lifetime is limited by the `refreshTokenAbsoluteLifetime` module parameter.

Configuring the OAuth2 client in Dex for connecting an application

This configuration is suitable for applications that can perform OAuth2 authentication themselves, without an OAuth2 proxy. The DexClient lets such applications use Dex.

```
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
spec:
  redirectURIs:
  - https://app.example.com/callback
  - https://app.example.com/callback-reserve
  allowedGroups:
  - Everyone
  - admins
  trustedPeers:
  - opendistro-sibling
```

In the authorization code flow, Dex accepts only the `S256` method of Proof Key for Code Exchange (PKCE). An application that sends a `code_challenge` with `code_challenge_method=plain`, or without `code_challenge_method` (the OAuth2 default is `plain`), is sent back to its redirect URI with the `invalid_request` error “Unsupported PKCE challenge method”. Such an application has to switch to `S256`. PKCE stays optional, and an application that sends no `code_challenge` signs in without it. The device flow (`/device/code`) is not affected by this check. Dex accepts `plain` for it, because the device code itself serves as the secret.

The DexClient group and email allow-lists are enforced on login and on every refresh-token request. Changing the lists denies further refresh for sessions that no longer match; those users must sign in again.

After the DexClient is created, Dex registers a client with the `dex-client-myname@mynamespace` ID (**clientID**).

The client access password (**clientSecret**) will be stored in the secret object:

```
apiVersion: v1
kind: Secret
metadata:
  name: dex-client-myname
  namespace: mynamespace
type: Opaque
data:
  clientSecret: c2VjcmV0
```

Granting an application access to the Kubernetes API

A DexClient or a DexAuthenticator can be allowed to obtain tokens that the Kubernetes API server accepts. To do that, set one of the following annotations on the object to `"true"`:

- `dexclient.deckhouse.io/allow-access-to-kubernetes` on a DexClient.
- `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` on a DexAuthenticator.

The following is a DexClient configuration example:

```
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec:
  redirectURIs:
  - https://app.example.com/callback
```

The annotation registers the client as a trusted peer of the privileged `kubernetes` OAuth2 client, allowing it to request ID tokens intended for the API server. In such a token, the username is determined by the `email` claim and groups by the `groups` claim. As a result, the application accesses the API server on behalf of the authenticated user and with the permissions granted to that user.

⚠ Access granted this way applies at the cluster level, even though DexClient and DexAuthenticator are namespaced objects. Therefore, only a subject allowed to `create` the `dexclients/allow-access-to-kubernetes` subresource (for DexClient) or the `dexauthenticators/allow-access-to-kubernetes` subresource (for DexAuthenticator) in the namespace of the object can add the annotation or change its value to `"true"`. The `d8:system-capability:user-authn:edit` role and the `ClusterAdmin` access level carry this permission. To grant it in one namespace, create a Role with the `create` verb on these subresources and bind it with a RoleBinding in that namespace. Permissions to create DexClient or DexAuthenticator objects are not sufficient.

Adding the annotation is restricted regardless of its value, including `"false"`. This is required for compatibility with Deckhouse Platform versions earlier than 1.77, where access is granted based on the presence of the annotation regardless of its value. If access to the Kubernetes API server is not required, do not add the annotation.

The restriction does not apply to resources that already have the annotation. A user with permissions to modify such a resource can continue to modify it, including removing the annotation or changing its value to disable access.

Limiting the providers of an application

By default, the sign-in page of an application offers every DexProvider of the cluster and the built-in users (User). To offer only some of them, list their names in `spec.allowedProviders` of the DexClient

or the DexAuthenticator. The name of a DexProvider is its `metadata.name`. The built-in users are `local`.

The following DexAuthenticator lets users sign in to the application only through the `corp-ldap` provider and as built-in users:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: admin-panel
  namespace: team
spec:
  applicationDomain: panel.example.com
  applicationIngressClassName: nginx
  allowedProviders:
  - corp-ldap
  - local
```

Dex hides the other providers on the sign-in page and refuses a sign-in through them even when the user opens the address of the provider directly. If the list holds one provider, the user goes straight to it, without the choice page. Dex also checks the provider at every refresh-token request and token exchange of the application, so a token issued through a provider that is no longer in the list stops refreshing, and its user has to sign in again.

The list works together with the other limits: `allowedEmails` and `allowedGroups` of the application, and `allowedIdentities` of the provider. A user has to pass all of them. An empty or missing list allows every provider. `trustedPeers` is the exception. A peer client can ask for a token whose audience is this client through a provider the peer itself allows, and this client's `allowedProviders` is not checked for that token. The same exception applies to `allowedEmails` and `allowedGroups`.

The users who signed in to the application through a provider that left the list have to sign in again even if the provider is added back. With `sessions.enabled`, Dex keeps the browser session of such a user. Once the provider is back in the list, the user can be signed in by that session without entering the credentials until the session ends.

A name that matches no DexProvider is accepted with a warning. Nobody can sign in through it until the provider is created. Therefore a DexAuthenticator or DexClient applied before its provider by a GitOps tool does not fail.

Exchanging tokens of other issuers

A TokenExchangeProvider lets Dex exchange a token of another issuer for its own token (RFC 8693 token exchange). The Kubernetes API server accepts the Dex token like the token of any other Dex user. Typical issuers are CI systems, the Dex of a neighbouring cluster and the ServiceAccount token issuer of another cluster.

Dex accepts a token if all of the following hold:

- The signature verifies with the keys of the issuer, from its discovery document (`keys.discovery`) or given in the object (`keys.jwks`).
- If the token header has `kid`, the signature is checked with the key of that `kid` only.
- The `iss` claim equals `issuer`, and the token has not expired.
- The token is valid for no longer than `subjectTokenMaxLifetime` (24 hours by default), counted from `iat` to `exp`, or from now to `exp` for a token without `iat`. The `exp` claim is required.
- The `aud` claim matches an entry of `audiences`.
- The claims listed in `claimValidation` carry the required values.
- The DexClient that sends the request is listed in `clients`.

The subject is named by `identity`. The user name is a prefix followed by the value of a claim, and groups are taken from claims in the same way. The API server takes the user name from the `email` claim. Without `identity.email`, the user name in the cluster is the result of `identity.username`; with `identity.email`, it is the result of `identity.email`. RBAC subjects use this name.

Example for users of a neighbouring cluster A. The Dex of cluster A issues the tokens for its DexClient `cluster-b` in the `d8-user-authn` namespace, so `aud` is `dex-client-cluster-b@d8-user-authn`. In cluster B, the DexClient `cluster-a-exchange` that exchanges them needs the `dexclient.deckhouse.io/allow-access-to-kubernetes` annotation. Apply the following manifest in cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: TokenExchangeProvider
metadata:
  name: cluster-a
spec:
  issuer: https://dex.cluster-a.example.com/
  keys:
    discovery:
      caData: |
        -----BEGIN CERTIFICATE-----
        <CA_OF_CLUSTER_A>
        -----END CERTIFICATE-----
  audiences:
  - exact: dex-client-cluster-b@d8-user-authn
  identity:
    username:
      claim: email
      prefix: "cluster-a:"
    groups:
    - claim: groups
      prefix: "cluster-a:"
  allowedIdentities:
    emailDomains:
    - example.com
  clients:
  - dexClient:
      name: cluster-a-exchange
      namespace: d8-user-authn
```

The client exchanges a token at the Dex token endpoint with `connector_id` set to `status.connectorID` of the object. An example for CI systems is in Access for CI/CD.

A TokenExchangeProvider has the following features:

- It serves token exchange only and is not shown on the sign-in page.
- A DexClient listed in `spec.clients` can exchange tokens through the provider even when its `allowedProviders` does not name the provider.
- A token of the issuer is accepted only as a JWT ID token. Access tokens are not accepted.
- A change of the object takes effect without a restart of Dex and applies to new exchanges. Dex tokens issued earlier stay valid until they expire. They are refused earlier if the provider loses its connector, that is, if the object is deleted or created again under the same name, or if the provider gives way to an earlier one (`PrefixConflict`). The refusal works under the same condition as for a deleted object, described below. After `PrefixConflict` is gone, the provider gets a connector with a new `status.connectorID`.
- A change refused as invalid (`InvalidSpec`, for example a malformed key or `caData` without a certificate) is not applied. The provider keeps working with its previous spec until the spec is fixed.
- Every prefix is required and ends with a colon. The prefix sets the subjects of the provider apart from people and from the subjects of other providers. Two providers cannot give the same names. A provider whose user name or group prefix overlaps one of a provider created earlier is not ready (`PrefixConflict`) and exchanges nothing until the conflict is removed.
- With `keys.discovery`, the provider becomes ready only after the discovery document and the keys of the issuer are read. Otherwise it is not ready (`DiscoveryFailed`) and is checked again every minute; its connector is written all the same, so a change that narrows the provider applies at once. The discovery document is read over https only.
- Deleting the object makes the Kubernetes API refuse every token issued through it within seconds if the `enableMultiTenancy` parameter of the `user-authz` module is enabled and the cluster runs Kubernetes 1.34 or later. Otherwise these tokens stay valid until they expire (the `idTokenTTL` parameter).
- User and group names starting with `peering:` are reserved for the ServiceAccounts of the `d8-peering` namespace. No DexProvider and no other TokenExchangeProvider can assert them.
- TokenExchangeProvider objects are managed by the `d8:system-capability:user-authn:edit` role, which the `d8:subsystem:iam:manager` role and the roles that include it carry, and in the basic role model by the `ClusterAdmin` access level. Whoever creates a TokenExchangeProvider chooses the issuer, its keys and the names, and so can get a token for any user or group outside the reserved prefixes. This right is equivalent to the right to impersonate users. Grant it to cluster administrators only. The module writes the Dex connectors and revocations in the `d8-user-authn` namespace; only a cluster administrator can change them.

Token exchange through a DexProvider is deprecated and off by default. An `OIDC` or `Gitlab` DexProvider exchanges tokens only with `spec.tokenExchange.enabled: true`. The parameter can only be turned off; nobody but the module can set `true`. During the upgrade from a release without this parameter, the module sets it once on the existing `OIDC` and `Gitlab` providers. It sets `true` if some DexClient has the `dexclient.deckhouse.io/allow-access-to-kubernetes` annotation, so that a cluster that may exchange tokens keeps doing it, and `false` otherwise. A DexAuthenticator with access to Kubernetes does not count. Its oauth2-proxy never exchanges tokens, and exchange with its client secret is not kept. The parameter and token exchange through a DexProvider will be removed after the next LTS release.

While a DexProvider exchanges tokens, the `D8UserAuthnDexProviderTokenExchangeDeprecated` alert fires. Move the exchange to a TokenExchangeProvider as described in Migration from DexProvider, then set `spec.tokenExchange.enabled: false`.

Browser sessions of Dex

By default, Dex does not remember a user between sign-ins: every application that sends the user to Dex gets the sign-in form. With the browser sessions of Dex enabled, Dex remembers the user signed in to the browser. To enable the sessions, set the `sessions.enabled` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    sessions:
      enabled: true
```

What changes with the sessions:

- The web interfaces of Deckhouse share the sign-in: the user enters the password and the second factor once and opens the others without them. These are the applications of the DexAuthenticator objects in the `d8-*` namespaces. User DexAuthenticator and DexClient objects do not share the sign-in. To turn the sharing off, set `sessions.sso` to `None`. The shared sign-in lasts while the browser uses any of these applications, up to `sessions.absoluteLifetime`. After `sessions.validIfNotUsedFor` without using them, the next application asks for the password again. A user of an external provider shares the sign-in while the user data Dex stored is fresh: after `sessions.identityRefresh.maxAge` without a sign-in or a refresh of the tokens of an application, the next application asks for the password again.
- The sign-in form shows the “Remember me” checkbox. With it, the session survives closing the browser.
- When a user signs out of an application of a DexAuthenticator, DexAuthenticator sends the browser to Dex, and Dex ends the session. Within seconds, the browser is signed out of every application of a DexAuthenticator it signed in to through this session. If the application and Dex are on different sites (different registrable domains), Dex asks the user to confirm the sign-out. An application signs out with a navigation of the browser: a script request (`fetch`, XHR) does not follow the redirect to Dex, and the session of Dex stays.
- Signing out of Dex does not sign the user out of an external OIDC provider: the provider may sign the user in again without a password. To end the session at the provider too, enable `upstreamLogout` of the provider, or set `promptType: login` to ask for the password at every sign-in.
- Dex reads the data of the user (such as the groups) from the provider again when the stored data is older than `sessions.identityRefresh.maxAge`. If the provider answers that the user no longer exists or is no longer allowed in, Dex ends all the sessions and tokens of the user of this provider.

What does not change:

- Tokens for `kubectl` (kubeconfig, `d8 login`) are issued only after an interactive sign-in, never from a session of the browser.
- An application of a DexAuthenticator stays signed in for `keepUsersLoggedInFor`, even when the session of Dex ends by its lifetime or idle timeout. Only signing out ends it.

How long a session lasts

A session of Dex ends at the first of these:

- `sessions.absoluteLifetime` after the sign-in, a week by default, however much the browser is

used;
- `sessions.validIfNotUsedFor` without use, a day by default;
- the sign-out of the user, or an end of the session by an administrator;
- the closing of the browser, if the user did not select “Remember me” at the sign-in.

The browser uses the session when it signs in to an application through it, and when an application signed in through it refreshes its tokens: an application of a DexAuthenticator object or of a DexClient object with `refreshTokenLifetime: Session`. Such an application refreshes its tokens every `idTokenTTL` while it is open and sends requests. A closed application, or one open in the browser of a sleeping computer, does not use the session.

With the defaults, a user who works every day enters the password and the second factor once a week, and once more after a break longer than a day, such as a weekend. The session survives a night.

When the session ends, the applications already open stay signed in for `keepUsersLoggedInFor`, but signing in to another web interface asks for the password again.

To ask for the password more often, shorten `sessions.validIfNotUsedFor`, for example to `1h`: the user then enters it again after every break of an hour. `sessions.validIfNotUsedFor` must not be longer than `sessions.absoluteLifetime`; when only `sessions.absoluteLifetime` is set and is shorter than a day, an unused session ends after `sessions.absoluteLifetime`.

Signing out of the application of a DexClient

An application of a DexClient object signs the user out of Dex by sending the browser to `https://dex.<PUBLIC_DOMAIN>/logout` with the `client_id` and `post_logout_redirect_uri` parameters. Dex returns the browser only to the addresses listed in `postLogoutRedirectURIs` of the DexClient:

```
https://dex.example.com/logout?client_id=dex-client-
myapp@mynamespace&post_logout_redirect_uri=https%3A%2F%2Fmyapp.example.com%2F
```

To learn that a session of a user ended, the application can set `backchannelLogoutURI`. Dex sends the notification once, without retries, when the user signs out of the session, the provider answers that the user is gone, a UserOperation or the self-service of the web interface ends the session, or the user is deleted; for a session ended outside Dex, within seconds. An application that must not miss the end can also check its tokens at the introspection endpoint of Dex (`/token/introspect`), which reports the tokens of an ended session or device inactive.

An application that requests offline access (refresh tokens) gets them only after an interactive sign-in. With `refreshTokenLifetime: Session`, the application can sign the user in from the session of the browser, and its tokens end when the user signs out of this session.

Local Authentication

Local authentication provides user verification and access management with support for configurable password policies, two-factor authentication (2FA), and group management.
In the module parameters, you can configure password complexity, password history, password rotation, lockout after failed sign-in attempts and per-IP rate limiting of failed attempts. No external authentication system is required.

Creating a user

The recommended way to create a local user is the `d8 iam user create` command. It supports interactive password entry, automatic password generation, group assignment, and time-to-live (TTL) for temporary users.

Examples:

Interactive password prompt (default when stdin is a terminal):

```
d8 iam user create anton --email anton@abc.com
```

Automatically generate a password (the generated password is shown in the command output once):

```
d8 iam user create anton --email anton@abc.com --generate-password
```

Read the password from stdin (for CI/CD pipelines):

```
echo "s3cret" | d8 iam user create anton --email anton@abc.com --password-stdin
```

Use a pre-computed bcrypt hash (e.g. from htpasswd):

```
d8 iam user create anton --email anton@abc.com --password-hash '$2y$10$abcdef...'
```

Create the user and add them to groups (auto-creating groups if missing):

```
d8 iam user create anton --email anton@abc.com --generate-password --member-of admins --
create-groups
```

Create a temporary user with a TTL:

```
d8 iam user create anton --email anton@abc.com --generate-password --ttl 24h
```

Preview the manifest without applying it:

```
d8 iam user create anton --email anton@abc.com --generate-password --dry-run -o yaml
```

As an alternative, you can create the User resource manually. Create a password and enter its hash encoded in base64 in the `password` field. The email address must be in lowercase.

Use the command below to calculate the password hash:

```
echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0;
echo
```

> **i** If the `htpasswd` command is not available, install the appropriate package:
>
> - `apache2-utils` — for Debian-based distributions.
> - `httpd-tools` — for CentOS-based distributions.
> - `apache2-htpasswd` — for ALT Linux.

Note that in the below example the `ttl` parameter is set.

```
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  # echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' |
base64 -w0; echo
  password:
'JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6SzhpeS5H'
  ttl: 24h
```

To make the user change the password at the first sign-in, set `requirePasswordChangeOnFirstLogin: true`. The field can be set only when the User is created.

Authorization rules grant permissions to users based on the email address from the issued token. Therefore, a User resource cannot be created if its `spec.email` value matches a `User` subject in an existing AuthorizationRule or ClusterAuthorizationRule resource. This prevents permissions from being unintentionally granted to a new user.

If the match is intentional, for example, if the authorization rule was created in advance, use a different email address or add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the User resource. The annotation acknowledges the name collision only. Assigning the grant still requires covering permissions or a can-assign range that includes the target roles.

Email addresses are converted to lowercase when matched, as this is how they are stored in the token. For example, `Admin@Example.com` matches the `admin@example.com` subject.

The restriction does not apply to existing users whose email addresses match subjects in authorization rules. Such users continue to work, and a warning is issued when a match is detected.

Consider this restriction when managing users and authorization rules declaratively. If a user and the

corresponding rule are created at the same time, the rule may be created first, causing creation of the User resource to be rejected. To allow such a match, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the User manifest.

Deleting a user does not automatically remove the corresponding subject from the authorization rule. As long as the subject remains in the rule, permissions continue to be granted to the specified email address, and a warning is issued that no corresponding user exists. If a user with the same email address is created later, that user receives these permissions. If this behavior is not required, remove the corresponding subject from the authorization rule.

Deleting a user

To delete a local user, use the `d8 iam user delete` command. By default, it also removes the user from all Group resources they belong to.

Examples:

Delete the user (and automatically remove them from all groups):

```
d8 iam user delete anton
```

Delete the user but keep their references in groups:

```
d8 iam user delete anton --keep-memberships
```

When a user is deleted, manually or after its `ttl` expires, the module ends the user's sessions in the same way as the `Logout` operation (see UserOperation lifecycle and side effects) and deletes the user's 2FA (TOTP) secret and lock in Dex. A user created again with the same name does not inherit the sessions, the 2FA secret or the lock.

Local user operations

Password reset, forced password change, 2FA reset, lock, unlock and logout operations are performed via the UserOperation resource. The `initiatorType` field indicates who initiated the operation: an administrator (`admin`) or the user (`self`).

> ⚠ Do not change `User.spec.password` after the User is created (for example, with `d8 k edit`). The field is immutable and does not update Dex credentials. Reset passwords with a UserOperation or `d8 iam user reset-password`.

Administrative operations

Use the `d8 iam user` commands for administrative actions on local users. They create a UserOperation resource with `initiatorType: admin`, wait for the operation to complete, and print the result.

You can delete or recreate a local user, or run a UserOperation (`ResetPassword`, `RequirePasswordChange`, `Reset2FA`, `Lock`, `Unlock`, `Logout`) against them, only if you can assign every role already granted to the user's email or groups (covering permissions or an explicit can-assign range). `initiatorType: self` does not bypass this check. The same rule applies to DexProvider objects. For details, see Limiting the identities of a provider.

After an administrator resets the password with `ResetPassword`, the user signs in with that password once. Dex then asks the user to set a new password that complies with the password policy. To keep the new password, set `resetPassword.requirePasswordChange: false` in the UserOperation.

Interactive password reset example:

```
d8 iam user reset-password admin
```

Example of reading the new password from stdin:

```
echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
```

Example of generating a new password automatically:

```
d8 iam user reset-password admin --generate-password
```

If the password is already hashed, pass the bcrypt hash without Base64 encoding:

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

2FA reset example:

```
d8 iam user reset2fa admin
```

Example of locking a user for 30 minutes:

```
d8 iam user lock admin 30m
```

The `d8 iam user lock` command locks a local user and accepts a duration in seconds, minutes or hours, for example `30s`, `10m` or `1h`. To lock a user for several days (`lock.for: 7d`) or permanently (`lock.for: permanent`), create a UserOperation as described in Creating UserOperation manually.

User unlock example:

```
d8 iam user unlock admin
```

By default, commands wait for the operation to complete. To only create a UserOperation and print its name, use the `--wait=false` flag.

Self-service operations

A user can sign out on all devices, and a local user can change their password, in the Deckhouse Platform web interface. The web interface creates a UserOperation with `initiatorType: self` for the user who is signed in:

- Signing out on all devices creates a `Logout` operation. It ends every session of the user, including kubeconfig files and the session in the web interface. Basic Authentication to the Kubernetes API through an LDAP provider keeps working with the password.
- Changing the password creates a `RequirePasswordChange` operation. At the next sign-in, Dex asks for the current password and a new one that complies with the password policy. The operation ends no sessions.

Changing the password is available only for local accounts (the built-in `Local` connector). Users who sign in through external authentication providers change their password in the corresponding system.

A user can also see the devices signed in to their account (the `status.sessions` field of their UserAccount) and sign out of them one by one:

- Signing out of one device creates a `Logout` operation with `logout.session` for a browser or `logout.grant` for another device, such as a kubeconfig file. The other devices keep their sessions. If the device is already signed out, for example its session has expired, signing out of it succeeds.
- Signing out of the other devices creates a `Logout` operation with `logout.exceptSession` or `logout.exceptGrant`. It keeps only the device the user acts from.

Signing out on all devices, signing out of the other devices or of another device, and changing the password are available only on a trusted device: one signed in at least `devices.trustedAfter` ago, 24 hours by default. A device signed in more recently can sign the user out only of itself. Someone who has just signed in with a stolen password therefore can neither sign the owner out nor change the password, while the owner sees the new device in the list on a trusted device and signs it out. A user without a trusted device asks an administrator: the operations of an administrator are not limited. With `devices.trustedAfter: 0s`, every device is trusted at once.

The age of a device counts from the first sign-in of its browser, not only the last one. At sign-in, Dex gives the browser a long-lived cookie and remembers when it first saw that browser sign in to the account (the `knownSince` field). A browser that signed out and signed in again keeps its age. Dex forgets the browser, and its next sign-in counts as new, when:

- the last sign-in of the browser is ended from another device or by an administrator. While the browser holds another active session or token of the account, it stays known;
- the user signs out of the other devices: every browser but the current one is forgotten, signed in or not;
- the user signs out on all devices, or an administrator locks the user, resets the password, resets 2FA, or signs the user out;

- the account has more remembered browsers than `devices.rememberedBrowsers` (20 by default): the browser seen longest ago is forgotten. With `0`, Dex remembers no browser, and every sign-in is a new device.

Someone who signs in with a stolen password uses a browser without that cookie, so the device is new for them. A stolen session is not covered: the age is that of the session or token of the request, so a copied cookie of Dex or of the web interface has the age of the owner's device.

If the password may have been stolen, act from a trusted device in this order:

1. Request a password change.
2. Right away, sign out on this device and sign in again: Dex asks for the current and the new password. The browser keeps its age.
3. Only then sign out of the other devices. The current browser stays known, so it stays trusted, and every other device, the intruder's included, is signed out.

Do steps 1 and 2 one right after the other. Until the new password is set, any sign-in asks for it, the intruder's with the stolen password included, and the age of the device is not checked on that page.

Do not start with signing out on all devices: it makes the current browser new as well, and for `devices.trustedAfter` it can neither request a password change nor sign out a device that signs in again with the stolen password.

A compromised password alone signs in to an account without 2FA. Enable 2FA with the `twoFactor` parameter.

Creating UserOperation manually

When the `d8 iam user` CLI is unavailable (for example, in CI/CD or automation scripts), create a UserOperation resource directly. Use `apiVersion: deckhouse.io/v1` and set `initiatorType: admin`.

Do not keep a UserOperation in a GitOps repository. The module deletes the operation after `retention.userOperations`, and an operation created again runs again.

Example — reset a local user's password (the `newPasswordHash` contains a bcrypt hash without Base64 encoding; the module encodes it automatically):

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: reset-password-admin
spec:
  user: admin
  type: ResetPassword
  initiatorType: admin
  resetPassword:
    newPasswordHash: "$2y$10$..."
```

Example — lock a local user for 1 hour:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: lock-admin-1h
spec:
  user: admin
  type: Lock
  initiatorType: admin
  lock:
    for: "1h"
```

Example — permanent lock:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: lock-admin-permanent
spec:
  user: admin
  type: Lock
  initiatorType: admin
  lock:
    for: "permanent"
```

Example — unlock:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: unlock-admin
spec:
  user: admin
  type: Unlock
  initiatorType: admin
```

Example — reset 2FA:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: reset-2fa-admin
spec:
  user: admin
  type: Reset2FA
  initiatorType: admin
```

Example — require a new password at the next sign-in without resetting the current one:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: require-password-change-admin
spec:
  user: admin
  type: RequirePasswordChange
  initiatorType: admin
```

Example — end every session of a user in every application and kubeconfig file:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: logout-admin
spec:
  user: admin
  type: Logout
  initiatorType: admin
```

Example — sign a user out of one device and keep the sessions on the others. Take the `id` of the device from `status.sessions` of the user's UserAccount (for details, see the Viewing account state section). For a device of the `Browser` kind, set it in `logout.session`; for a device of the `Grant` kind, such as a kubeconfig file, in `logout.grant`:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: logout-admin-kubeconfig
spec:
  user: admin
  type: Logout
  initiatorType: admin
  logout:
    grant: qx3w2ttiyq5ovzvbuehdcbbse
```

If the user has no such session or refresh token, for example the session has expired or the user has signed out of it, the operation ends nothing and completes with the `Succeeded` status and the `the user has no such session` or `the user has no such grant` message. The operation does not check whether the ID belongs to another user: the session of another user gives the same result and stays active. Take the ID from the UserAccount of the user set in `spec.user` or `spec.target`.

Operations on external users

For users of external providers of any type (LDAP, Atlassian Crowd, OIDC, SAML, GitHub, GitLab, Bitbucket Cloud), use the `spec.target` field instead of `spec.user`. The `Lock`, `Unlock`, `Logout` and `Reset2FA` operation types are supported for external users. Only the users of the LDAP and Atlassian Crowd providers can have a second factor. For a user of a provider of another type, `Reset2FA` only ends the sessions.

The operation finds the user’s account by the email the provider reported at the user’s last sign-in. The `connectorID` and `userID` fields narrow the search down to one provider and one user ID. A lock applies to one pair of a provider and a user. It blocks only the account found and does not block a local user or a user of another provider with the same email. A local user is addressed by `spec.user` only. Dex checks the lock at every sign-in through the provider, at every refresh-token request, at token exchange, and at Basic Authentication to the API server through an LDAP provider. For a Crowd or OIDC provider, basic-auth-proxy checks the credentials with the provider directly, so the lock does not apply to Basic Authentication.

The operation fails if no account has the email (the user has never signed in) or if more than one account matches. In that case, set `connectorID`, and `userID` if the provider has several users with the email.

A lock applies to the user ID that the provider returns at sign-in, so this ID must stay the same at every sign-in of the user. For a SAML provider, set `nameIDPolicyFormat` to `persistent` or `emailAddress`, or leave it unset (Dex then requests `persistent`). With `transient`, the provider returns a new ID at every sign-in. A lock then does not apply to the next sign-in of the user, and once the user has signed in twice, the operation fails because more than one user of the provider has the email.

Example — lock an external user by `connectorID` + email for 30 minutes:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: lock-external-user
spec:
  target:
    connectorID: my-ldap
    email: jane.doe@example.org
  type: Lock
  initiatorType: admin
  lock:
    for: "30m"
```

Users of a deleted provider

If you delete a DexProvider and create it again under the same name within 30 days, the second factor and the lock of its users are kept. If you create it later, the 2FA of a user who has not signed in for 30 days is registered again at the first sign-in. A user whose lock is still in force keeps the lock and the second factor until the lock ends.

UserOperation lifecycle and side effects

Using a UserOperation object has the following specifics:

- A UserOperation is a **single-use** object. After creation, the module processes it and writes the result to `status.phase` (`Succeeded` or `Failed`).
- Operations are **automatically deleted** after `retention.userOperations` (24 hours by default).
- A UserOperation runs once. Changes to its spec after creation are ignored.
- To perform a new action, create a new UserOperation.

⚠ The `Lock`, `ResetPassword`, `Reset2FA` and `Logout` operations end the sessions of the user at once:

- Dex refuses every refresh token, authorization code and browser session it issued to the user before the operation. Applications and kubeconfig files ask the user to sign in again.
- DexAuthenticator stops accepting the sessions of the user within seconds.
- The Kubernetes API refuses the ID tokens issued to the user before the operation within seconds if the `enableMultiTenancy` parameter of the `user-authz` module is enabled and the cluster runs Kubernetes 1.34 or later. Otherwise these ID tokens stay valid for the Kubernetes API until they expire (the `idTokenTTL` parameter).
- Basic Authentication to the Kubernetes API through an LDAP provider checks the password with Dex again. After `Logout`, it keeps working with the password.

`Unlock` and `RequirePasswordChange` end no sessions. Only `Reset2FA` removes the user's 2FA secret. A `Lock` refuses sign-in, refresh-token requests and token exchange until the lock ends. The `status.revocation` field of the operation shows what it revoked.

Checking operation status

To check the operation status, follow these steps:

1. Get the list of all operations:

   ```
   d8 k get useroperations
   ```

2. Get the full status of an operation:

   ```
   d8 k get useroperation <name> -o yaml
   ```

3. Get only the completion status:

   ```
   d8 k get useroperation <name> -o jsonpath='{.status.phase}'
   ```

Viewing account state

The module keeps a UserAccount object for each local user and for each user who has signed in through an external provider. It shows when the user last signed in, whether the user has set up 2FA, whether the account is locked, until when, and how many failed login attempts were counted. Do not create or edit UserAccount objects.

To list the accounts, run:

```
d8 k get useraccounts
```

The `status.sessions` field lists the devices signed in to the account, the most recently active first: browsers signed in to Dex (the `Browser` kind) and devices holding a refresh token not bound to a live browser session of Dex, such as kubeconfig files (the `Grant` kind). For each device, it shows the applications, the client address and User-Agent of the last sign-in, the time of the sign-in, the hour of the last activity, and when the session of the device ends unless the device is used again (`endsAt`). The client address is right only when the Ingress controller sees it (for details, see the Which client address does Dex see? section). To view the devices of an account, run:

```
d8 k get useraccount <NAME> -o jsonpath='{range .status.sessions[*]}{.kind}{"\t"}{.id}
{"\t"}{.clients}{"\t"}{.ipAddress}{"\t"}{.lastActivity}{"\t"}{.endsAt}{"\n"}{end}'
```

While the browser sessions of Dex are disabled, every sign-in to an application of a DexAuthenticator is a device of its own.

The maximum number of devices on which a user can remain logged in to a single application at the same time is set by the `devices.maxPerApplication` parameter (default: 10). A browser counts as one device for all domains of the application, while each kubeconfig file counts as a separate device. If a user logs in from a new device and the limit in `devices.maxPerApplication` has already been reached, the session is terminated on the device that has been unused the longest. Trusted devices (a device becomes trusted after the time specified in the `devices.trustedAfter` parameter has elapsed) are disconnected last.

Automatic lockout

If the number of failed sign-in attempts exceeds `passwordPolicy.lockout.maxAttempts`, Dex locks the user for `lockout.lockDuration` and then unlocks them automatically. For LDAP and Crowd users, this works only if the provider type is listed in `lockout.applyToConnectors`. Dex does not create a UserOperation for this lockout. To unlock the user earlier, run `d8 iam user unlock` or create a UserOperation of type `Unlock`.

While a local, LDAP or Crowd user is locked, Dex refuses a password sign-in whether the password is correct or not: with the `429` status code and the “Account temporarily locked” page after an automatic lockout, with the `403` status code and the “Account locked by an administrator” page while a lock set by an administrator is in force. The password change page refuses a locked user in the same way. An automatic lockout does not shorten a lock that ends later, including a lock set by an administrator.

Adding a user to a group

Users can be grouped to manage access rights. The recommended way to manage groups is the `d8 iam group` command.

Examples:

Create a group:

```
d8 iam group create admins
```

Add a user to a group:

```
d8 iam group add-member admins user anton
```

Add a nested group:

```
d8 iam group add-member admins group devops
```

Remove a user from a group:

```
d8 iam group remove-member admins user anton
```

Delete a group:

```
d8 iam group delete admins
```

As an alternative, you can create the Group resource manually. Example manifest of the Group resource for a group:

```
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin
```

Where `members` is a list of members: `kind: User` with `name` = `User.metadata.name`, or a nested `kind: Group` with `name` = `Group.spec.name` (the name in the token, not `metadata.name`).

Nested groups must not form a cycle. A change to a Group resource is rejected if after it a group becomes its own member, directly or through other groups. If the hierarchy already contains a cycle, changes that do not create a new one are accepted with a warning that shows the path of the cycle, where each group is a member of the previous one. To break the cycle, remove one of these groups from the members of the previous group.

The group name is stored in the issued token without modification. It is indistinguishable from a group name received from an external authentication provider. Therefore, a Group resource cannot be created if its `spec.name` value matches a `Group` subject in an existing AuthorizationRule or ClusterAuthorizationRule resource. This prevents permissions from being unintentionally granted to members of a new group.

If the match is intentional, use a different group name or add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the Group resource.

Unlike email addresses, group names are not converted to lowercase when matched. Therefore, group names that differ only in letter case are considered different.

The restriction does not apply to existing groups whose names match subjects in authorization rules. Such groups continue to work, and a warning is issued when a match is detected.

Consider this restriction when managing groups and authorization rules declaratively. If a group and the corresponding rule are created at the same time, the rule may be created first, causing creation of the Group resource to be rejected. To allow such a match, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the Group manifest.

Deleting a group does not automatically remove the corresponding subject from the authorization rule. As long as the subject remains in the rule, permissions continue to be granted to a group with the specified name. If a group with the same name is created later, its members receive these permissions. If this behavior is not required, remove the corresponding subject from the authorization rule.

Viewing users and groups

Use the `d8 iam get` and `d8 iam list` commands to view users, groups, and their effective access (groups, grants, and access level).

Examples:

List all users with effective access:

```
d8 iam list users
```

Details for a specific user (groups, grants, access level):

```
d8 iam get user anton
```

List all groups:

```
d8 iam list groups
```

Details for a specific group (members, grants):

```
d8 iam get group admins
```

Password policy

Password policy settings allow controlling password complexity, rotation, and user lockout:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"
```

Field description:

- `complexityLevel`: Password complexity level. One of `None`, `Low`, `Fair`, `Good`, `Excellent` or `Custom`.
- `custom`: Custom complexity rules (used only with `complexityLevel: Custom`):
  - `custom.minLength`: Minimum number of characters in a password.
  - `custom.specialCharacters`: If `true`, require at least one special character.
  - `custom.numbers`: If `true`, require at least one digit.
  - `custom.capitalized`: If `true`, require at least one uppercase letter.
  - `custom.repeatedChars`: If `true`, forbid more than 2 identical characters in a row.
- `passwordHistoryLimit`: Number of previous passwords stored in the system to prevent their reuse.
- `lockout`: Lockout settings after exceeding the limit of failed login attempts:
  - `lockout.maxAttempts`: Limit of allowed failed login attempts.
  - `lockout.lockDuration`: User lockout duration.
  - `lockout.applyToConnectors`: External provider types the lockout also applies to: `LDAP`, `Crowd`. Local users are always covered.
- `rotation`: Password rotation settings:
  - `rotation.interval`: Period for mandatory password change.

To limit failed login attempts per client IP address, use the `rateLimit` parameter.

When a local user signs in with a password that has to change, Dex opens the password change page instead of the application. A password has to change after an administrator resets it with `ResetPassword`, when it does not meet the complexity rules, and when it is older than `rotation.interval`. The page asks only for the new password, twice, because the sign-in form has just checked the current one. If 2FA is enabled, Dex first asks for the code and opens the password change page only after it, so someone who knows only the password cannot change it. After the change, the sign-in continues as usual and opens the application.

The password change page opens only during the sign-in that led to it and in the same browser. If the page is opened in another browser or after the sign-in has expired, Dex does not open it, and the user signs in again.

Example with a custom complexity rules:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 3
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Custom
      custom:
        minLength: 10
        specialCharacters: true
        numbers: false
        capitalized: true
        repeatedChars: false
      passwordHistoryLimit: 10
```

Two-factor authentication (2FA)

2FA increases security by requiring a code from a `TOTP` authenticator application (for example, Google Authenticator) or a security key (`WebAuthn`) during login. It can be required from the local users and from the users of the LDAP and Atlassian Crowd providers.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 4
  enabled: true
  settings:
    twoFactor:
      enabled: true
      issuerName: "awesome-app"
      providerTypes: [Local, LDAP]
```

Field description:

- `enabled`: Enables or disables 2FA.
- `issuerName`: Name displayed in the authenticator application when adding an account (with `WebAuthn`, in the dialog of the browser that registers the key).
- `methods`: The second factors: `TOTP` (the default), `WebAuthn`, or both, see Several methods.
- `require`: How two `methods` combine. Required with two methods, not allowed with one.
- `providerTypeOverrides`: Other `methods` for one type of provider, see Several methods.
- `webAuthn`: Parameters of the `WebAuthn` method, see Security keys.
- `providerTypes`: Types of the providers whose users have to pass the second factor: `Local` (the built-in users), `LDAP` and `Crowd`. The users of every provider of a listed type are asked for it. The OIDC, SAML, GitHub, GitLab and Bitbucket Cloud providers are not on the list. The identity provider handles the second factor for them.

The `twoFactor` parameter replaces `staticUsers2FA`, which covered the local users only. A ModuleConfig of version 3 with `staticUsers2FA` keeps working, because the module converts it.

> **i** After enabling 2FA, each user must register in the authenticator application during their first login. Until then, anyone who knows the password can register their own device, so ask the users to sign in right after you turn 2FA on.

Dex asks the user for the code (or the key) at a sign-in through a provider of a listed type, except while a browser session of Dex is valid. Sign-ins through the other providers go without it. Turning 2FA on for a provider type does not end the refresh tokens or the Dex browser sessions issued before. A refresh token keeps issuing tokens without the code until it expires (by default, up to `refreshTokenAbsoluteLifetime`, 4380h), and a browser session created before the change passes without the code. After enabling 2FA, run a UserOperation of type `Logout` for the users of the providers

of the listed types.

Basic Authentication to the Kubernetes API (`enableBasicAuth` of an LDAP or Crowd provider) has no page for a code, so it is not possible for a provider of a listed type. The module refuses the settings that turn 2FA on while such a provider exists, and a provider that turns `enableBasicAuth` on while 2FA covers its type. To use both, give the Kubernetes API users a kubeconfig file with the web interface instead of Basic Authentication.

Security keys (WebAuthn)

With `methods: [WebAuthn]`, Dex asks the user to touch a security key (for example, a YubiKey) or to confirm the sign-in with the Touch ID or Windows Hello of the device, instead of entering a code. The key confirms a sign-in only for the domain it is bound to, so a fake sign-in page on another domain gets no response it can reuse.

The following settings ask the users for a security key:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 4
  enabled: true
  settings:
    twoFactor:
      enabled: true
      methods: [WebAuthn]
      issuerName: "Example"
      webAuthn:
        rpID: example.com
```

The method works as follows:

- A key is bound to the domain set by the `rpID` parameter. By default, it is the domain of the Dex address (`dex.<PUBLIC_DOMAIN>`). To keep the keys through a change of `publicDomainTemplate`, set `rpID` to the stable parent domain of the Dex address, for example `example.com` for `dex.example.com`. A change of `rpID`, or of `publicDomainTemplate` while `rpID` is not set, makes every registered key unusable. A user with such a key can neither sign in nor register a new one. Reset the second factor of the user with a UserOperation of type `Reset2FA`, and the user registers a key again at the next sign-in.
- Dex accepts a security key of any make and model and does not check which key it is.
- The browser has to support WebAuthn.
- The user registers the key at the first sign-in after the method is turned on, as the user of TOTP registers the application. The maximum number of keys that a user can have is specified in the `webAuthn.maxCredentials` parameter (3 by default). A user adds one with the “Add a key” button on the page that confirms a sign-in with a key, after a registered key has confirmed it. Until the registration, anyone who knows the password can register their own key, so ask the users to sign in right after the switch.
- A single key cannot be removed. If any of the keys has been lost or compromised, reset the user’s `WebAuthn` method with a `Reset2FA` operation. Until then, such a key still confirms a sign-in for whoever knows the password. The operation removes every key of the user, and the user registers the keys again at the next sign-in.
- When the methods change, the TOTP secret and the keys the user had stay until a `Reset2FA`, so asking for a method again brings them back.

Several methods

If two methods are specified in the `methods` parameter, the `require` parameter defines how they are combined. With `Any`, the user passes one of them. At the first sign-in the user chooses the method and registers it. Later Dex asks for a registered method, the one used last first, and a user who has registered both can sign in another way. A user who has registered one method cannot register the other at a sign-in, because otherwise anyone who knows the password could add a method of their own. To let a user switch, remove the registered method with a `Reset2FA` operation. With `All`, the user passes both, the code first and then the key. A user who has only one of them confirms the sign-in with it first and only then registers the other.

The following settings let each user choose a code or a key:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 4
  enabled: true
  settings:
    twoFactor:
      enabled: true
      methods: [TOTP, WebAuthn]
      require: Any
      webAuthn:
        rpID: example.com
```

A Deckhouse Platform version without `require: Any` refuses such settings. Before a rollback to it, switch `require` to `All` or keep one method.

`providerTypeOverrides` sets other methods for one type of provider from `providerTypes`, for example a key for the users of the LDAP providers and a code for the local users:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 4
  enabled: true
  settings:
    twoFactor:
      enabled: true
      methods: [TOTP]
      providerTypes: [Local, LDAP, Crowd]
      providerTypeOverrides:
      - providerType: LDAP
        methods: [WebAuthn]
      - providerType: Crowd
        methods: [TOTP, WebAuthn]
        require: All
      webAuthn:
        rpID: example.com
```

A type without an entry takes `methods` and `require`. Each type appears once, and only a type listed in `providerTypes`.

The second factor of an account

UserAccount shows in `status.twoFactor` the `methods` and `require` asked of the users of its type of provider, and in `enrolled` whether the account has them. `totp` and `webAuthn` show the TOTP secret and the number of keys the account has, including those of a method no longer asked for, which do not count for `enrolled`.

To make a user register the second factor again, for example after losing the device, reset the user's 2FA. For a local user, use `d8 iam user reset2fa <USERNAME>` or a UserOperation of the `Reset2FA` type with `spec.user`. For a user of an external provider, use a UserOperation of the `Reset2FA` type with `spec.target`. Without other parameters the operation removes every method. To remove one, list it in `spec.reset2FA.methods`:

```
apiVersion: deckhouse.io/v1
kind: UserOperation
metadata:
  name: reset-key-jane
spec:
  type: Reset2FA
  initiatorType: admin
  user: jane
  reset2FA:
    methods: [WebAuthn]
```

The operation also ends the sessions of the user. Other operations, including `Lock` and `ResetPassword`, keep the second factor.

Assigning permissions to a user or group

To grant permissions to a user or a group, bind one of the `user-authz` roles with a RoleBinding or ClusterRoleBinding, using the user's email or the group name as the subject. See the granular role-based model.

> **i** **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The user-authn module: FAQ

> **i** The module lifecycle stage: General Availability

How to secure my application?

To enable Dex authentication for your application, follow these steps:

1. Create a DexAuthenticator.

   In the namespace of the DexAuthenticator, the module creates the `<NAME>-dex-authenticator` Service, an Ingress if `applicationIngressClassName` is set and an HTTPRoute if `gatewayAPI` is set. To check that the object is served, see Checking that a DexAuthenticator is served.

   Example of the DexAuthenticator custom resource:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  # Name of the DexAuthenticator. The name of its Service is `app-name-dex-
authenticator`.
  name: app-name
  # Namespace of the application.
  namespace: app-ns
spec:
  # Your application's domain. Requests to it will be redirected for Dex
authentication.
  applicationDomain: "app-name.kube.my-domain.com"
  # A parameter that determines whether to send the `Authorization: Bearer`
header to the application.
  # This one is useful in combination with auth_request in NGINX.
  # If sendAuthorizationHeader is true, add Authorization to the
nginx.ingress.kubernetes.io/auth-response-headers annotation of the
application's Ingress or to the alb.network.deckhouse.io/auth-response-headers
annotation of its HTTPRoute.
  sendAuthorizationHeader: false
  # The name of the Secret containing the SSL certificate.
  applicationIngressCertificateSecretName: "ingress-tls"
  # The name of the Ingress class to use in the Ingress resource created for the
Dex authenticator.
  applicationIngressClassName: "nginx"
  # The duration of the active user session.
  keepUsersLoggedInFor: "720h"
  # The list of groups whose users are allowed to authenticate.
  allowedGroups:
  - everyone
  - admins
  # The list of addresses and networks for which authentication is allowed.
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
```

2. Connect your application to Dex.

   To do this, add annotations to the resource through which the app is published. The set of annotations depends on how the application is published. Choose the relevant option:

- Through an Ingress resource
- Through ALBInstance or ClusterALBInstance

Add the following annotations to the application’s Ingress resource:

- `nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-authenticator/sign_in`
- `nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-Request-Email`
- `nginx.ingress.kubernetes.io/auth-url: https://<SERVICE_NAME>.<NS>.svc.{{ C_DOMAIN }}/dex-authenticator/auth`, where:
  - `SERVICE_NAME`: Name of the authenticator’s Service. Usually, it is `<NAME>-dex-authenticator` (`<NAME>` is the `metadata.name` of the DexAuthenticator).
  - `NS`: Value of the `metadata.namespace` parameter of the DexAuthenticator.
  - `C_DOMAIN`: Cluster domain (the `clusterDomain` parameter of the `control-plane-manager` module).

i If the DexAuthenticator `<NAME>` is too long, the Service name may be truncated. To find the correct service name, use the following command (specify the namespace name and DexAuthenticator name):

```
d8 k get service -n <NS> -l "deckhouse.io/dex-authenticator-for=<NAME>" -o
jsonpath='{.items[0].metadata.name}'
```

Example of annotations for an application’s Ingress resource for connecting to Dex:

```
annotations:
  nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-authenticator/sign_in
  nginx.ingress.kubernetes.io/auth-url: https://app-name-dex-authenticator.app-
ns.svc.cluster.local/dex-authenticator/auth
  nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-
Request-Email
```

An application that specifies only `applicationIngressClassName` gets no HTTPRoute. If Ingress publication is turned off with `global.modules.ingress.enabled`, it gets no Ingress either. The application itself stays reachable through the route you publish it with, but none of its `/dex-authenticator` endpoints are, so the redirect to sign in lands on the application and no one can log in. Other applications are unaffected.

If the application is published through an ALBInstance or ClusterALBInstance resource (for details, see the `alb` module documentation), add the following annotations to the application’s HTTPRoute resource:

- `alb.network.deckhouse.io/auth-signin: https://<application-domain>/dex-authenticator/sign_in` — unlike nginx, the `alb` controller does not support the `$host` variable, so the application’s domain must be specified explicitly.
- `alb.network.deckhouse.io/auth-response-headers: X-Auth-Request-User,X-Auth-Request-Email`.
- `alb.network.deckhouse.io/auth-url: https://<SERVICE_NAME>.<NS>.svc.<C_DOMAIN>/dex-authenticator/auth`, where `SERVICE_NAME`, `NS`, and `C_DOMAIN` are determined the same way as for an Ingress resource.

Example of annotations for an application’s HTTPRoute resource for connecting to Dex:

```
annotations:
  alb.network.deckhouse.io/auth-signin: https://app-name.kube.my-domain.com/dex-
authenticator/sign_in
  alb.network.deckhouse.io/auth-url: https://app-name-dex-authenticator.app-
ns.svc.cluster.local/dex-authenticator/auth
  alb.network.deckhouse.io/auth-response-headers: X-Auth-Request-User,X-Auth-Request-
Email
```

Also, in the DexAuthenticator resource, specify the same ListenerSet used to publish the application’s domain:

```
spec:
  gatewayAPI:
    applicationHTTPRouteListenerSetName: my-listenerset
```

The HTTPRoute created for the DexAuthenticator attaches to that same ListenerSet, so the ListenerSet must accept routes from the namespace of the DexAuthenticator.

An application that specifies only `gatewayAPI` gets no Ingress. If Gateway API is turned off with `global.modules.gatewayAPI.enabled`, or the cluster serves no HTTPRoute API, it gets no HTTPRoute either, with the same result: the application stays reachable, its `/dex-authenticator` endpoints do not, and no one can log in. Other applications are unaffected.

⚠ When setting `sendAuthorizationHeader: true`, list all necessary headers in the Ingress (or in the HTTPRoute, if the `alb` module is used) in the corresponding annotation, since the `Authorization` header is not transmitted by default.

For details on what is passed in the `Authorization` header and how to specify it in the annotation, see How to pass the user’s login and groups to an application .

⚠ The application Ingress must have TLS configured. DexAuthenticator does not support HTTP-only Ingress resources.

Setting up CIDR-based restrictions

To allow authentication only from specific networks, list them in `whitelistSourceRanges`. The limit applies to the `/dex-authenticator` endpoints. To restrict access to the application itself, or to let some networks in without authentication, use the annotations of the application's Ingress:

- To restrict access by IP and keep Dex authentication, add the annotation with a comma-separated list of allowed CIDRs:

  ```
  nginx.ingress.kubernetes.io/whitelist-source-range: 192.168.0.0/32,1.1.1.1
  ```

- To allow access without Dex authentication for users from specified networks while requiring authentication for others, add the annotation:

  ```
  nginx.ingress.kubernetes.io/satisfy: "any"
  ```

Checking that a DexAuthenticator is served

The module starts serving a new or changed DexAuthenticator within a few minutes. Until then, requests to the application get an error.

To see whether the module serves the DexAuthenticator on every domain it names, look at the `Served` column of the command output:

```
d8 k -n <NAMESPACE> get dexauthenticators
```

If the column shows `False`, the `reason` and `message` fields of the `Served` condition say why. The reasons are listed in the description of the `status.conditions` field. To print the condition, run:

```
d8 k -n <NAMESPACE> get dexauthenticator <NAME> -o jsonpath='{.status.conditions[?
(@.type=="Served")]}'
```

Here `<NAMESPACE>` is the namespace of the DexAuthenticator and `<NAME>` is its name.

One namespace may have about 600 DexAuthenticator objects. Limit their number with a ResourceQuota:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dexauthenticators
  namespace: <NAMESPACE>
spec:
  hard:
    count/dexauthenticators.deckhouse.io: "20"
```

A DexAuthenticator that stays not served raises the `D8DexAuthenticatorNotServed` alert after 10 minutes.

Updating from a version with a dex-authenticator for every DexAuthenticator

After the update, the users of every DexAuthenticator sign in once more. A DexAuthenticator the module cannot serve keeps its previous dex-authenticator, and changes of `allowedGroups` and `allowedEmails` do not apply to it. The `reason` field of the `Served` condition tells the cause.

How to pass the user's login and groups to an application?

By default DexAuthenticator passes only two headers to the application: `X-Auth-Request-User` (based on the opaque `sub` claim) and `X-Auth-Request-Email`. The user's groups are not passed as a header. It grows unboundedly with the number of groups, so it is disabled in DP, and it cannot be enabled.

To give the application full information about the user, including groups, enable `sendAuthorizationHeader`:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-ns
spec:
  applicationDomain: "app-name.kube.my-domain.com"
  applicationIngressClassName: "nginx"
  applicationIngressCertificateSecretName: "ingress-tls"
  sendAuthorizationHeader: true
```

In this case, the application receives the `Authorization: Bearer <id_token>` header, where `<id_token>` is a JWT signed by Dex (for details on the token contents and how to process it in your application, see JWT token contents and processing considerations).

Regardless of how the application is published, this header is not passed to the application automatically. It must be explicitly listed in the annotation of the resource through which the application is published. Choose the relevant option depending on how the application is published:

- Through an Ingress resource
- Through ALBInstance or ClusterALBInstance

If the application is published through an Ingress resource (for details, see the `ingress-nginx` module documentation), when setting `sendAuthorizationHeader: true`, you need to:

- Specify the headers to pass to the application in the `nginx.ingress.kubernetes.io/auth-response-headers` annotation.
- Also increase the buffer size using the `nginx.ingress.kubernetes.io/proxy-buffer-size` annotation, since a JWT with many groups does not fit into the default buffer.

Example of specifying the headers and buffer size using the corresponding annotations:

```
annotations:
  nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-authenticator/sign_in
  nginx.ingress.kubernetes.io/auth-url: https://app-name-dex-authenticator.app-
ns.svc.cluster.local/dex-authenticator/auth
  nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-
Request-Email,Authorization
  nginx.ingress.kubernetes.io/proxy-buffer-size: 32k
```

If the `Authorization` header is not listed in `auth-response-headers`, the application won’t receive it. If `proxy-buffer-size` is not increased, requests will fail with a 500 error, and the Ingress controller log will show `upstream sent too big header while reading response header from upstream`.

If the application is published through an ALBInstance or ClusterALBInstance resource (for details, see the `alb` module documentation), when setting `sendAuthorizationHeader: true`, specify the headers to pass to the application in the `alb.network.deckhouse.io/auth-response-headers` annotation of the HTTPRoute resource:

```
annotations:
  alb.network.deckhouse.io/auth-signin: https://app-name.kube.my-domain.com/dex-
authenticator/sign_in
  alb.network.deckhouse.io/auth-url: https://app-name-dex-authenticator.app-
ns.svc.cluster.local/dex-authenticator/auth
  alb.network.deckhouse.io/auth-response-headers: Authorization
```

For the `alb.network.deckhouse.io/auth-response-headers` annotation, it is enough to specify only `Authorization`, since it already passes the base set of headers by default.

Also, in the DexAuthenticator resource, specify the same ListenerSet used to publish the application’s domain:

```
spec:
  gatewayAPI:
    applicationHTTPRouteListenerSetName: my-listenerset
```

JWT token contents and processing considerations

Example of the JWT payload for a static user (the User and Group resources):

```
{
  "iss": "https://dex.kube.my-domain.com/",
  "sub": "Cg1qb2huLmRvZUBleGFtcGxlEgVsb2NhbA",
  "aud": "dex-authenticator-app-name@app-ns",
  "exp": 1757000600,
  "iat": 1757000000,
  "email": "john.doe@example.com",
  "email_verified": true,
  "name": "john-doe",
  "preferred_username": "",
  "groups": ["everyone", "developers"]
}
```

When parsing the token in your application, keep in mind the following:

- Use the `email` field to identify the user. The `sub` field is opaque (it is not predictable and cannot be used as a meaningful user identifier outside the context of a specific system), and `preferred_username` is empty for static users (external authentication providers may populate it).
- The `name` field contains the object's name (from the `metadata.name` field of the User object), not the user's display name.
- The `aud` field contains the authenticator client identifier (`dex-authenticator-<name>@<namespace>`), not your application's own OIDC client identifier. An application that validates `aud` against its own `client_id` will reject such a token.
- Verify the signature with the JWKS endpoint `https://<DEX_DOMAIN>/keys`, where `<DEX_DOMAIN>` is the `publicDomainTemplate` value with `%s` replaced by `dex`.
- The token's lifetime is defined by the `idTokenTTL` parameter (10 minutes by default). DexAuthenticator refreshes the token on its own, so the application always gets a current one.

If the application supports OIDC on its own, use the DexClient resource instead of DexAuthenticator: the application will request the required scope itself and get a `refresh_token` in addition to the `id_token`.

Authentication flow with DexAuthenticator

DexAuthenticator sets its cookies with the `Secure` attribute, so they are sent only over HTTPS.

1. Dex redirects the user to the provider's login page in most cases and waits for the user to be redirected back to the `/callback` URL. However, some providers like LDAP or Atlassian Crowd do not support this flow. The user must enter credentials in the Dex login form instead, and Dex will validate them by making a request to the provider's API.
2. DexAuthenticator starts the sign-in with Proof Key for Code Exchange (PKCE) using the `S256` method and gets the tokens and the signing keys from the `dex` Service in the `d8-user-authn` namespace. It checks the certificate of Dex against the certificate authority (CA) of the internal Dex certificate.
3. DexAuthenticator sets the cookie with the full refresh token (instead of issuing a ticket as for the ID token) because Redis does not persist data. If no ID token is found in Redis by the ticket, the user can request a new ID token by providing the refresh token from the cookie.
4. DexAuthenticator sets the `Authorization` HTTP header to the ID token value from Redis. This is not required for services like `upmeter`, as `upmeter` permissions are less granular. For the Kubernetes Dashboard, it is critical: the Dashboard passes the ID token on to access the Kubernetes API.

Error 500 while Dex is unavailable

While Dex does not answer, DexAuthenticator cannot refresh a session whose ID token has expired. It keeps the session for one `idTokenTTL` period and answers the requests of the session with the `503` status code. If the application is published through an Ingress, the user sees the error page of the Ingress controller with the `500` status code. Reload the page after Dex starts answering: the session keeps working. If Dex does not answer for longer than the period, the session ends, and the user signs in again.

Handling an expired session in a single-page application

Without a session, a request to the application is redirected to the sign-in page from the `auth-signin` annotation (`/dex-authenticator/sign_in`). DexAuthenticator sends the user to Dex only when the browser navigates to that page. A request with the `Sec-Fetch-Mode: navigate` header is a navigation. If the browser does not send `Sec-Fetch-Mode`, a request without the `Accept: application/json` and `X-Requested-With: XMLHttpRequest` headers is a navigation. A script request (`fetch`, XHR) gets the `401` status code with the `{}` body instead of a redirect to Dex, which the browser would block as a cross-origin request.

To send the user to sign in, reload the page when a script request gets `401`:

```
const response = await fetch("/api/items");
if (response.status === 401) {
  window.location.reload();
}
```

A sign-in that took longer than 10 minutes

The system stores the temporary login state (a one-time authentication request) for 10 minutes. If the user does not complete it within this time (for example, due to slow entry of the login/password on the external provider's page, a long wait for the code during MFA, a lengthy recovery of a forgotten password, etc.), the login must be started over.

Ending a DexAuthenticator session

DexAuthenticator ends the sessions of a user within seconds after a `Lock`, `ResetPassword`, `Reset2FA` or `Logout` operation and after the user is deleted. A request with an ended session gets the same answer as a request without a session.

If DexAuthenticator does not receive the revocations, the `D8UserAuthnRevocationFeedNotConsumed` alert fires after 15 minutes.

When a user signs out of an application, DexAuthenticator revokes the refresh token of the session in Dex. A copy of the session cookie stops working too.

With the browser sessions of Dex enabled, DexAuthenticator then sends the browser to Dex to end the session of Dex, and the browser is signed out of the other applications it signed in to through this session. Sign the user out with a navigation, not with a script request, which does not follow the redirect to Dex:

```
window.location.assign("/dex-authenticator/sign_out");
```

How to generate a kubeconfig and access Kubernetes API?

To generate a kubeconfig, refer to the «How to generate a kubeconfig to access the Kubernetes API?» section of the `control-plane-manager` module documentation.

Configuring kube-apiserver

Using the `control-plane-manager` module, DP passes `kube-apiserver` an AuthenticationConfiguration with a JWT authenticator for Dex: issuer `https://<DEX_DOMAIN>/` (where `<DEX_DOMAIN>` is the `publicDomainTemplate` value with `%s` replaced by `dex`), audience `kubernetes`, the username from the `email` claim and the groups from the `groups` claim. If `controlPlaneConfigurator.dexCAMode` provides a CA, it is embedded in the configuration. Usernames and groups that start with `system:` are rejected.

The flow of accessing Kubernetes API with generated kubeconfig

1. `kube-apiserver` reads the Dex configuration (issuer and JWKS endpoint). In Kubernetes 1.30 and later, it reads it through the Dex Service inside the cluster.
2. The kubeconfig generated in the Deckhouse web UI contains the ID token and the refresh token.
3. After receiving a request with an ID token, `kube-apiserver` verifies that the token is signed by the provider configured in step 1 using keys from the JWKS endpoint. It then compares the token's `iss` and `aud` claim values with the configuration.

How to rotate the secret of the kubernetes OAuth2 client?

The secret of the privileged `kubernetes` OAuth2 client is stored in the `kubernetes-dex-client-app-secret` Secret of the `d8-user-authn` namespace. The same value is used by the `kubeconfig-generator`, `kubeconfig-generator-<index>`, `kubeconfig-publish-api` and `kubeconfig-<slug>` OAuth2 clients, and by basic-auth-proxy when basic authentication is enabled for an LDAP provider.

Deleting the Secret does not rotate the value: while it is still present in the module's values, the owning hook renders the very same one back.

To rotate the secret, do the following:

1. If a GitOps tool controls the `d8-user-authn` namespace, exclude the `kubernetes-dex-client-app-secret` Secret from syncing. Otherwise the GitOps tool will restore the previous value.
2. Empty the `secret` field:

   ```
   d8 k -n d8-user-authn patch secret kubernetes-dex-client-app-secret --type merge -p '{"data": {"secret":""}}'
   ```

3. Restart DP so that the hook picks the empty field up and generates a new value:

   ```
   d8 k -n d8-system rollout restart deployment/deckhouse
   ```

4. Verify that the secret value changed:

   ```
   d8 k -n d8-user-authn get secret kubernetes-dex-client-app-secret -o jsonpath='{.data.secret}'
   ```

   If it did not, repeat the steps 2 and 3. The module may have restored the previous value before DP was restarted.

Once the secret has been rotated, the configuration of components that use it will be updated automatically and the corresponding pods will be restarted.

Kubeconfig files generated in the Deckhouse web UI before the rotation carry the old client secret and stop refreshing tokens. Generate them again. ID tokens issued before the rotation keep working until they expire, which is configured via `settings.idTokenTTL` (10 minutes by default).

How to enable Kerberos (SPNEGO) SSO for LDAP?

If clients run in a corporate SSO environment (browser trusts the Dex host), Dex can accept Kerberos tickets via `Authorization: Negotiate` and log in without the password form.

Enabling Kerberos (SPNEGO) SSO for LDAP:

1. In AD/KDC, create/provision an SPN `HTTP/<DEX_FQDN>` for a service account and generate a `keytab`.
2. In the cluster, create a Secret in the `d8-user-authn` namespace with the `krb5.keytab` data key.
3. In the LDAP DexProvider resource, enable `spec.ldap.kerberos`:
   - `enabled: true`
   - `keytabSecretName: <secret name>`
   - optional: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`

Dex will mount the `keytab` automatically and start accepting SPNEGO. A server-side `krb5.conf` is not required — tickets are validated using the keytab.

How to configure Basic Authentication for accessing Kubernetes API via LDAP?

Use the `control-plane-manager` module settings:

1. Enable the `apiserver.publishAPI` parameter in the `control-plane-manager` module configuration.
2. Create a DexProvider of type `LDAP` and set `enableBasicAuth: true`.
3. Grant the LDAP groups access with a RoleBinding or ClusterRoleBinding of the granular role-based model.
4. Provide users with a kubeconfig configured for Basic Authentication (LDAP username and password).

> ⚠ Only one authentication provider in the cluster can have `enableBasicAuth` enabled.

A detailed example is described in the Usage section.

How is Dex protected against credential brute-forcing?

Dex limits failed sign-in attempts per client IP address on the sign-in form, the token endpoint, the password change page and the 2FA steps. Configure the limit with the `rateLimit` parameter. It is turned off by default.

Dex also locks an account after failed sign-in attempts. Configure the lockout with the `passwordPolicy.lockout` parameter. It applies to local users, and to LDAP and Crowd users if their types are listed in `applyToConnectors`.

Which client address does Dex see?

Dex uses the address of the client in three places:

- the limit of failed attempts per address (`rateLimit`);
- the sign-in log (the `client_remote_addr` key);
- the devices of a user (`ipAddress` in the UserAccount object).

Dex takes the address that the Ingress controller of the platform passes on (see `rateLimit` for the headers it trusts). So the address is right only when the Ingress controller sees the address of the client itself. A balancer in front of the Ingress controller that replaces the address of the client with its own hides it. Then Dex sees the address of the balancer for every user:

- the limit counts the failed attempts of all users together, so the failures of one user can make Dex refuse the sign-ins of the others;
- the sign-in log and the devices of a user show the address of the balancer.

To pass the address of the client to the Ingress controller, configure the IngressNginxController object:

- if the balancer supports the PROXY protocol, use an inlet with it, such as `LoadBalancerWithProxyProtocol` or `HostPortWithProxyProtocol`;
- if the requests come through an HTTP proxy that adds the `X-Forwarded-For` header, enable `behindL7Proxy` for the inlet and list the addresses of the proxy in `acceptClientIPHeadersFrom`. The `realIPHeader` parameter selects another header and works only with `behindL7Proxy`.

To check the address Dex sees, sign in and compare the `client_remote_addr` key of the `login successful` line in the Dex log with the address of your computer.

How to find out why a sign-in was refused?

Dex logs every sign-in in JSON. A refused sign-in is logged at the `WARN` level with the `reason` field:

- `invalid_credentials`: The login or the password is wrong, or the password change page was opened outside the sign-in that led to it.
- `invalid totp`: The 2FA code is wrong or has already been used.
- `not_in_allowed_groups`: The user is not in the groups that the provider settings require.
- `not allowed`: The user is not on an allow-list. The `rule` field names the list.
- `account_locked`: The user is locked.
- `rate_limited`: Too many failed attempts came from the client IP address. The line contains the requested `path` instead of the provider, the client and the user.
- `upstream_refused`: The provider refused to extend the session because the user no longer exists in the provider or has lost access.
- `revoked`: The refresh token, the authorization code or the browser session was issued before a `Lock`, `ResetPassword`, `Reset2FA` or `Logout` operation or before the user was deleted.
- `login_expired`: The sign-in request expired or was not found (the sign-in already finished, or it never existed, including a forged or replayed `state`). The line contains the requested `path` and, if the request path contains the provider ID, `connector_id` instead of the client and the user.

The line also contains `connector_id`, `client_id` and the known user attributes, such as `username` and `email`. A group refusal also contains `user_groups` and `allowed_groups`. A failure of Dex or of the provider is logged at the `ERROR` level.

To find the refusals in the Dex logs, run:

```
d8 k -n d8-user-authn logs -l app=dex -c dex --tail=-1 | grep '"reason"'
```

A UserOperation is in the Failed status — what should I do?

Check the `status.message` field of the UserOperation for the error description:

```
d8 k get useroperation <name> -o jsonpath='{.status.message}'
```

Fix the cause (for example, an invalid password hash or a non-existent user), then create a new UserOperation. A UserOperation runs once, and changes to its spec after creation are ignored.

How do I unlock a user?

Use the command:

```
d8 iam user unlock <USERNAME>
```

The command unlocks a local user. To unlock a user of an external provider, create a UserOperation with `type: Unlock` and `spec.target` as described in Operations on external users. A UserOperation with `spec.user` unlocks a local user as well. The `Lock` operation has already ended the sessions of the user, and `Unlock` does not restore them, so the user signs in again. Only `Reset2FA` removes the user's 2FA secret.

A user was locked automatically — why?

The number of failed login attempts exceeded `passwordPolicy.lockout.maxAttempts`. The user is locked for the duration specified in `passwordPolicy.lockout.lockDuration` and is unlocked automatically afterwards. Local users are always covered. LDAP and Crowd users are locked only if their type is listed in `passwordPolicy.lockout.applyToConnectors`. Dex stores the lock itself and does not create a UserOperation. An administrator can also unlock the user manually with `d8 iam user unlock <USERNAME>` (local users) or by creating a UserOperation with `type: Unlock` (with `spec.user` for local users, with `spec.target` for LDAP and Crowd users).

The `Locked` and `LockedUntil` columns of `d8 k get useraccounts` show whether a user is locked and until when. The module keeps a UserAccount object for each local user and for each user who has signed in through an external provider.

Can I delete a UserIdentity object?

No. Dex keeps the 2FA secret, the lock and the last sign-in of an account in a UserIdentity object in the `d8-user-authn` namespace. The object also holds the generation of the account in the `user-authn.deckhouse.io/generation` annotation, which the module uses to revoke the account's tokens. The module refuses a request to delete a UserIdentity. To remove a local user, delete the User. To reset 2FA or a lock, use a UserOperation of the `Reset2FA` or `Unlock` type.

Can I cancel a UserOperation?

No. A UserOperation runs once. To reverse its effect, create an opposite operation, for example a UserOperation with `type: Unlock` after a `Lock` operation.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The user-authz module

> Available with limitations in: **Open/CE**
>
> Available without limitations in: **Core, BE, SE, SE+, Ultimate/EE**

> The module lifecycle stage: General Availability

The module generates role-based access model objects based on the standard Kubernetes RBAC mechanism. The module creates a set of cluster roles (ClusterRole) suitable for most user and group access management tasks.

⚠ The module implements two role-based models: the granular one (recommended) and the basic one, built around the ClusterAuthorizationRule and AuthorizationRule resources (its support will be discontinued in future releases).

The models are not resource-compatible — automatic conversion is impossible — but they can be used at the same time: the permissions of both models are summed up.

The bindings that grant users the access levels of ClusterAuthorizationRule and AuthorizationRule are reconciled by the `user-authz-controller` component: it watches the rules and keeps their ClusterRoleBindings and RoleBindings in sync, and reports the result in the `status` of each rule (`d8 k get clusterauthorizationrules` shows the `READY` and `BINDINGS` columns). A rule with an unsupported `accessLevel` gets `Ready=False` with the `InvalidSpec` reason instead of breaking the module. The controller exports metrics on the rules it sees, the bindings they need, the bindings that exist, and the drift between them. The module ships alerts on controller unavailability, persistent drift, and rules whose bindings are not applied (for details, see the FAQ).

Granular role-based model

Unlike the basic Deckhouse Platform role-based model, the granular role-based one does not use ClusterAuthorizationRule and AuthorizationRule resources. Access rights are configured in the standard Kubernetes RBAC way: by creating RoleBinding or ClusterRoleBinding resources and specifying one of the roles prepared by the `user-authz` module in them. To grant access to all namespaces of a project at once, use the ProjectRoleBinding and ClusterProjectRoleBinding resources of the `multitenancy-manager` module.

i Access does not have to be granted by hand-writing YAML manifests: the Deckhouse web UI provides an access grant wizard. It walks you through the steps (who gets access → where → at which level), picks the right binding kind itself (RoleBinding, ClusterRoleBinding, ProjectRoleBinding, or ClusterProjectRoleBinding), and lets you assemble a custom role from ready-made building blocks without writing YAML.

The module creates special aggregated cluster roles (ClusterRole). By using these roles in RoleBinding or ClusterRoleBinding, you can do the following:

- Manage access to modules of a specific subsystem.

  For example, you can use the `d8:subsystem:network:manager` role in ClusterRoleBinding to allow a network administrator to configure *network* modules (such as `cni-cilium`, `ingress-nginx`, `istio`, etc.).

- Manage access to *user* resources of modules within namespaces.

  For example, the `d8:namespace:manager` role in RoleBinding enables deleting/creating/editing the PodLoggingConfig resource in the namespace. At the same time, it does not grant access to the cluster-wide ClusterLoggingConfig and ClusterLogDestination resources of the `log-shipper` module, nor does it allow configuration of the `log-shipper` module itself.

Role scopes

Every role operates in one of four scopes. The scope defines *where* the granted permissions apply and *which resource* is used to assign the role:

Scope	Role name format	Intended for	Which resource is used to assign the role
Namespace	`d8:namespace:<level>`	Application users (developers)	RoleBinding in a specific namespace.
Project	`d8:project:<level>`	Teams working with projects	ProjectRoleBinding or ClusterProjectRoleBinding for the whole project, RoleBinding for one namespace of the project.

Scope	Role name format	Intended for	Which resource is used to assign the role
Subsystem	`d8:subsystem:<subsystem>:<level>`	Administrators of a part of the platform	ClusterRoleBinding.
Whole platform	`d8:system:<level>`	Platform administrators	ClusterRoleBinding.

The roles created by the module are divided into the following classes:

- Namespace roles — for assigning rights to users (such as application developers) **in a specific namespace**.
- Project roles — for assigning rights **in all namespaces of a project at once**.
- System and subsystem roles — for assigning rights to platform administrators and administrators of a part of the platform.

Access levels

Each scope provides several access levels for roles:

- The “namespace” and “project” scopes have five levels: `viewer` → `user` → `manager` → `admin` → `superadmin`.
- The “subsystem” and “whole platform” scopes have three levels: `viewer` → `manager` → `superadmin`. There are no `user` and `admin` levels here: the system level has no “user” resources that could be used without administering them.

Access levels form a hierarchy with a cumulative principle: each next level inherits all permissions of the previous one and extends them. For example, the `manager` level in the namespace scope includes everything the `user` level allows, which in turn includes everything the `viewer` level allows.

Using ClusterAuthorizationRule and AuthorizationRule together with RBAC

If multitenancy mode is enabled in the cluster (via `enableMultiTenancy: true`), the effective permissions of a user are the combination of all permissions granted by the following sources:

- **Permissions granted by ClusterAuthorizationRule**: Applied only inside the namespaces allowed by its `namespaceSelector` parameter (or the deprecated `limitNamespaces` parameter).
- **Permissions granted by AuthorizationRule**: Applied inside the namespace the AuthorizationRule is created in.
- **Permissions granted by ordinary RoleBinding**: Applied inside the namespace the RoleBinding is created in.
- **Permissions granted by ordinary ClusterRoleBinding** that aren’t generated by the `user-authz` module: Applied cluster-wide.

The namespace restrictions set in a ClusterAuthorizationRule apply only to the permissions granted by that ClusterAuthorizationRule. They do not cancel permissions granted via RoleBinding, ClusterRoleBinding, or AuthorizationRule in other namespaces. Likewise, the permissions granted by ClusterAuthorizationRule do not extend to those namespaces.

The namespace restrictions are enforced by the authorization webhook, so they do not apply to the identities that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. A ClusterAuthorizationRule that names such a ServiceAccount, directly or through a group such as `system:serviceaccounts`, grants its access level in every namespace. Create the ServiceAccounts of external systems in ordinary namespaces.

For example, if a user has a ClusterAuthorizationRule with `accessLevel: Editor` limited to the `ns-a` namespace and a RoleBinding with the `view` role in the `ns-b` namespace, they get the `Editor` permissions in the `ns-a` namespace and read-only permissions in the `ns-b` namespace. Permissions granted by the RoleBinding aren’t restricted by the ClusterAuthorizationRule, and the `Editor` level set in the ClusterAuthorizationRule doesn’t apply to the `ns-b` namespace.

Namespace roles

> ⚠ A namespace role cannot be assigned through a ClusterRoleBinding. Use a RoleBinding in a namespace, or a ProjectRoleBinding or ClusterProjectRoleBinding for all namespaces of a project.

Namespace roles are intended to assign rights to a user **in a specific namespace**. Users refer to, for example, developers who use a cluster configured by an administrator to deploy their applications. Such users don’t need to manage Deckhouse Platform modules or a cluster, but they need to be able to, for

example, create their Ingress resources, configure application authentication, and collect logs from applications.

The namespace role defines permissions for accessing namespaced resources of modules and standard namespaced resources of Kubernetes (Pod, Deployment, Secret, ConfigMap, etc.).

The module creates the following namespace roles:

- `d8:namespace:viewer` — allows viewing standard Kubernetes resources (except for Secrets and RBAC resources), Gateway API resources, pod logs and metrics in a specific namespace, as well as authenticating in the cluster;
- `d8:namespace:user` — in addition to the role `d8:namespace:viewer` it allows viewing secrets in a specific namespace, connecting to pods (`kubectl exec`, `kubectl attach`), deleting pods (but not creating or modifying them), executing `kubectl port-forward` and `kubectl proxy`, as well as changing the number of replicas of controllers;
- `d8:namespace:manager` — in addition to the role `d8:namespace:user` it allows managing module resources (for example, Certificate, PodLoggingConfig, etc.) and standard namespaced Kubernetes resources (Pod, Deployment, ConfigMap, Secret, Service, Ingress, NetworkPolicy, CronJob, etc.), as well as Gateway API resources (Gateway, ListenerSet, HTTPRoute, GRPCRoute, ReferenceGrant, etc.) in a specific namespace;
- `d8:namespace:admin` — in addition to the role `d8:namespace:manager` it allows managing the resources ResourceQuota, LimitRange, ServiceAccount, Role, RoleBinding in a specific namespace;
- `d8:namespace:superadmin` — in addition to the role `d8:namespace:admin` it allows security-sensitive operations: minting ServiceAccount tokens and making requests on behalf of ServiceAccounts.

The detailed split of permissions between `admin` and `superadmin` is described in the section Admin level restrictions and superadmin rights.

Automatic access to cluster-wide catalog resources

Working in a namespace requires reading some cluster-wide “catalogs”: for example, to set `storageClassName` or `ingressClassName` in a manifest, one needs to see the list of StorageClasses and IngressClasses. Therefore, every subject that receives a RoleBinding to any `d8:namespace:*` or `d8:project:*` role automatically also gets read access to such catalog resources (StorageClass, IngressClass, PriorityClass, RuntimeClass, VolumeSnapshotClass, ClusterLogDestination, etc.).

Technically this appears as an automatically created ClusterRoleBinding to the `d8:dict` role labelled `rbac.deckhouse.io/dict: "true"`. These objects are managed by the platform: they appear when the subject gets its first namespace binding and are removed when the subject has none left — no manual editing is needed.

Project roles

> ⚠ A project role cannot be assigned via ClusterRoleBinding — such an attempt is rejected. To assign a role across a whole project, use ProjectRoleBinding or ClusterProjectRoleBinding. A plain RoleBinding in one of the project namespaces is also allowed — the role then applies in that namespace only.

Project roles (`d8:project:<level>`) are intended for working with projects — isolated environments that may span several namespaces. The levels are the same as for namespace roles: `viewer`, `user`, `manager`, `admin`, `superadmin`.

Each project role includes all permissions of the namespace role of the same level and additionally grants permissions to manage the project itself:

- `d8:project:viewer` — the permissions of `d8:namespace:viewer` plus viewing the project’s ProjectNamespace and ProjectRoleBinding resources;
- `d8:project:user` — the permissions of `d8:namespace:user` plus viewing the project’s ProjectNamespace and ProjectRoleBinding resources;
- `d8:project:manager` — the permissions of `d8:namespace:manager` plus managing the project’s additional namespaces (ProjectNamespace resources);
- `d8:project:admin` — the permissions of `d8:namespace:admin` plus managing access to the project (ProjectRoleBinding resources) and the right to bind the built-in `d8:project:*` and `d8:namespace:*` roles (except the `superadmin` level) to other users within the project;
- `d8:project:superadmin` — analogous to the relation between `d8:namespace:superadmin` and `d8:namespace:admin`.

A role assigned via ProjectRoleBinding automatically applies in **all** namespaces of the project — the main one as well as the additional ones, including those created later.

System and subsystem roles

> ⚠ The system and subsystem roles do not grant access to the namespace of user applications.
>
> They grant access only to system namespaces (starting with `d8-` or `kube-`), and only to those system namespaces where the modules of the corresponding role subsystem are running.
>
> Whether a role lets you grant access to other people is described in Who can assign roles.

System and subsystem roles are intended for assigning rights to manage the entire platform or a part of it (the subsystem), but not the users applications themselves. The subsystem role, for example, can allow a security administrator to manage security modules (responsible for the security functions of the cluster). Thus, the security administrator will be able to configure security policies, certificate issuance, etc., but will not be able to manage other cluster functions (such as access management, network and monitoring settings) or change settings in the namespaces of users applications.

> ⚠ A system/subsystem role limits which modules and namespaces a subject can address, but it does not limit the privileges a subject can obtain through the modules it is allowed to manage. This matters most for the `iam` subsystem, because the right to manage authentication and authorization is equivalent to full control over the cluster.
>
> A subject that can manage the `user-authn` module can register an identity provider or reset the credentials of any local user. A subject that can manage the `user-authz` module can modify authorization rules. In both cases, they can obtain an identity with any privileges, including cluster administrator, so treat an `iam` subsystem role as a cluster administrator role when planning access.

The system/subsystem role defines access rights:

- to cluster-wide Kubernetes resources;
- to manage Deckhouse Platform modules (ModuleConfig resource) within the subsystem of the role, or to all Deckhouse Platform modules for the role `d8:system:*`;
- to manage cluster-wide resources of Deckhouse Platform modules within the subsystem of the role, or to all resources of Deckhouse Platform modules for the role `d8:system:*`;
- to read the namespaces of the modules of the role's subsystem, or of all Deckhouse Platform modules for the role `d8:system:*`; writing there is reserved for the `superadmin` level (see Access of system and subsystem roles to the namespaces of the modules).

The role name format is `d8:system:<ACCESS_LEVEL>` for the system roles and `d8:subsystem:<SUBSYSTEM>:<ACCESS_LEVEL>` for the subsystem roles, where:

- `SUBSYSTEM` is the role's subsystem;
- `ACCESS_LEVEL` is the access level.

  Examples:

  - `d8:system:viewer` — access to view the configuration of all Deckhouse Platform modules (ModuleConfig resource), their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except Secrets and RBAC resources) in the namespaces of all Deckhouse Platform modules;
  - `d8:system:manager` — similar to the role `d8:system:viewer`, but with the right to view/create/modify/delete the configuration of all Deckhouse Platform modules (ModuleConfig resource) and their cluster-wide resources; in the namespaces of all Deckhouse Platform modules it keeps the read access of `d8:system:viewer`;
  - `d8:subsystem:observability:viewer` — access to view the configuration of Deckhouse Platform modules (ModuleConfig resource) from the `observability` area, their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except secrets and RBAC resources) in the system namespaces `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, `kube-prometheus-pushgateway`.

The module provides three access levels for system and subsystem roles:

- `viewer` — allows viewing standard Kubernetes resources, the configuration of modules (ModuleConfig resources), cluster-wide resources of modules, and namespaced resources of modules in the module namespace;
- `manager` — in addition to the level `viewer` it allows managing the configuration of modules (ModuleConfig resources) and the cluster-wide resources of modules; in the module namespaces it grants the same read access as `viewer`;
- `superadmin` — in addition to the level `manager` it allows everything in the module namespaces: managing standard Kubernetes resources and the namespaced resources of modules, reading

secrets, connecting to pods, issuing ServiceAccount tokens and managing the system resources of the subsystem modules.

The `d8:subsystem:iam:manager` role grants the following permissions for UserOperation objects.

Resource or subresource	Verbs
`useroperations`	`create`, `get`, `list`, `watch`
`useroperations/sensitive`	`get`, `list`, `watch`

The `useroperations/sensitive` permission allows reading `spec.resetPassword.newPasswordHash` while the field is present in a UserOperation.

Access of system and subsystem roles to the namespaces of the modules

The holder of a system or subsystem role automatically gets a RoleBinding to a namespace role in every system namespace in which the modules of the role's subsystem run (a `d8:system:*` role covers all system namespaces): the `viewer` and `manager` levels get `d8:namespace:viewer`, the `superadmin` level gets `d8:namespace:superadmin`. The platform creates and removes these RoleBindings itself (they carry the label `rbac.deckhouse.io/automated: "true"`), one per ClusterRoleBinding of the role.

The namespaces of the modules belong to the platform. They hold ServiceAccounts with cluster-wide rights (the Deckhouse controller among them) and the credentials of the modules, so any write there, and any read of a Secret there, is in effect the right to act as those ServiceAccounts anywhere in the cluster. This is why a `manager` configures its subsystem through ModuleConfig objects and the cluster-wide resources of the modules, and in the module namespaces only looks: objects, logs, metrics and events, but no secrets, no `kubectl exec`, `attach` or `port-forward`, no deleting or scaling of pods and no creating of anything. A `superadmin` can do all of that. Grant this level deliberately and to a named subject: it is the emergency path into the platform, and its actions are visible in the audit log. Do not use impersonation of system users (`--as`) as a substitute.

Keep in mind:

- A RoleBinding to `d8:namespace:user` or a higher level in a system namespace, created by hand, grants the subject the rights of every ServiceAccount of that namespace. The same holds for the `allowAccessToSystemNamespaces: true` option of a ClusterAuthorizationRule of the previous role model: with it, the `Admin` and `ClusterAdmin` access levels are equivalent to a cluster administrator.
- The system namespaces are those the multitenancy-manager module never turns into projects: `d8-*`, `kube-*`, `upmeter-*`, `default` and the namespaces labelled `heritage: deckhouse` or `heritage: upmeter`.
- A custom system or subsystem role (`d8:custom:*`) is projected as `viewer` whatever its `rbac.deckhouse.io/use-role` label says: the webhook refuses a higher value, and a role created with one earlier is projected as `viewer` with a `ProjectionCapped` event on the role.
- For module authors: the system capability of a module (the one aggregated into the `manager` role of its subsystem) must not grant write access to namespaced objects of the module's namespace that produce workloads with a ServiceAccount, tokens or identities (pods and pod controllers, Secrets, ServiceAccounts, Roles and RoleBindings): such a right would hand the `manager` the ServiceAccounts of the module. The `dmt` linter will check this for the `rbac.yaml` of a module.

Who can assign roles

A system or subsystem role does not, by itself, let you grant access to other people.

Creating a User or Group that is not already a grant subject is ordinary object creation.

Creating a User for an email that already carries a grant, writing or deleting a ClusterAuthorizationRule, or resetting that identity through UserOperation, is granting roles. The request is admitted only if the requester already has covering permissions or is explicitly allowed to assign those roles.

Connecting a DexProvider is measured the same way against the identities the provider can assert: `spec.allowedIdentities` (emails, email domains, groups) and the provider's own group filters bound that set, and only the roles already granted inside it count. A provider without an email limit can assert every email, including the ones that hold SuperAdmin, so writing it requires SuperAdmin. Basic Authentication of an OIDC or a Crowd provider bypasses Dex and its limits, so such a provider counts as one without an email limit, and an OIDC one also without a group limit; a Crowd provider keeps only `crowd.groups` on the group axis. Details and the list of changes admitted without a check are in the user-authn module documentation.

The ClusterAuthorizationRule `spec.accessLevel` field is a basic-model level: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`. An `iam` subsystem manager and a holder of `d8:system:manager` assign the levels `ClusterEditor` includes: `User`, `PrivilegedUser`, `Editor` and `ClusterEditor`, but not `Admin`, `ClusterAdmin` or `SuperAdmin`, and neither the

Kubernetes `cluster-admin` ClusterRole.

A ClusterAuthorizationRule that opens the system namespaces counts as a `ClusterAdmin` assignment whatever its `accessLevel`, so these managers cannot write it: a rule with `allowAccessToSystemNamespaces: true`, with `namespaceSelector.matchAny`, with a `namespaceSelector.labelSelector` that does not name its namespaces by the `kubernetes.io/metadata.name` label, a rule for a subject that another rule already lets into them (the rules of a subject add up), or any rule while multitenancy mode is disabled. Without multitenancy mode, as in CE, these managers therefore cannot write ClusterAuthorizationRule objects at all.

The rules of a request add up: those of its user, of every group in its token and of its ServiceAccount. A rule therefore also opens the system namespaces when another rule already lets in one of the requests it applies to: a local user it names or a member of its groups at any depth of nesting, with the groups that user is in and the groups above them; the requester, with the groups of its own token; any request, through `system:authenticated` (a ServiceAccount also through `system:serviceaccounts` and `system:serviceaccounts:<namespace>`).

A rule for `system:authenticated` applies to every request, so it opens the system namespaces as soon as any other rule opens them. A rule for `system:serviceaccounts` does so as soon as another rule lets in a ServiceAccount, and a rule for `system:serviceaccounts:<namespace>` as soon as another rule lets in a ServiceAccount of that namespace. A User subject named `system:serviceaccount:<namespace>:<name>` is that ServiceAccount.

A member of a Group gets the rules of every group above it, so writing a Group needs the range for their roles too. For the same reason, these managers cannot add a member that another rule already lets into the system namespaces to a Group that has a ClusterAuthorizationRule, or is in a group that has one. The groups an identity provider asserts for other users are unknown to the webhook and are not taken into account.

For the same reason a user or a group that such a rule names counts as a `ClusterAdmin` target: these managers cannot create a User for its email, add members to such a Group, run a UserOperation on such a user or connect a DexProvider that can assert such an identity. Turning multitenancy mode off in the `user-authz` ModuleConfig opens the system namespaces to every rule and requires the `ClusterAdmin` range as well. Within the granular model the IAM manager assigns the `iam` subsystem roles up to `d8:subsystem:iam:manager`, and the system manager assigns the roles of any scope up to the `manager` level.

These managers do not write admission webhooks and policies (ValidatingWebhookConfiguration, MutatingWebhookConfiguration, ValidatingAdmissionPolicy and ValidatingAdmissionPolicyBinding): the superadmins of the `security` and `cluster` subsystems and `d8:system:superadmin` do. A mutating webhook can change any object the platform creates, its own pods included, and removing a webhook or a policy switches off what it enforces, such as the limits above. The ValidatingWebhookConfigurations `d8-deckhouse-validating-webhook-handler-*`, which hold the webhooks of the Deckhouse modules, can be created, changed or deleted only by webhook-handler, the platform components and the superadmins, including the holders of a role that grants every verb on everything.

A requester with every permission across the cluster counts as SuperAdmin in these checks and can assign any role, including `SuperAdmin` and `cluster-admin`. For example, a subject of the Kubernetes `cluster-admin` ClusterRole has such permissions if the role is granted by a ClusterRoleBinding or by a ClusterAuthorizationRule without `namespaceSelector` and `limitNamespaces`. If multitenancy mode is enabled, such a rule reaches the system namespaces only with `namespaceSelector.matchAny: true` (or the deprecated `allowAccessToSystemNamespaces: true`). Permissions from a ClusterAuthorizationRule with `namespaceSelector` or `limitNamespaces` do not count toward this, even if the rule grants `cluster-admin`. Any non-empty `limitNamespaces` counts as a limit on purpose, even if its entries match every namespace, such as `.*` together with `allowAccessToSystemNamespaces: true`. Such a rule still counts through its `accessLevel`. For example, `accessLevel: SuperAdmin` gives the SuperAdmin range whatever the namespace limits of the rule.

`limitNamespaces` and `namespaceSelector` restrict access to namespaced resources only. A subject with the `ClusterAdmin` or `SuperAdmin` access level can still create ClusterRoleBinding and ClusterAuthorizationRule objects, so these fields do not confine what it can grant across the cluster. Give these levels only to subjects that may hold access to the whole cluster. A ClusterAuthorizationRule counts only through the bindings that `user-authz-controller` has created for it.

Webhook deny messages call the `accessLevel` values basic, so they are not confused with the granular role levels (`viewer` / `manager` / `superadmin`).

Permission to create User and Group objects in the `user-authn` module is not enough: it does not grant the roles already attached to that email.

Global resource dictionaries

In addition to the roles and capabilities of the granular model, the module creates a special ClusterRole `d8:dict`. It grants **read-only** access to cluster-scoped “reference” resources that users commonly need to discover when creating objects — for example, a user creating a PersistentVolumeClaim needs to see

the available StorageClasses, and a user creating an Ingress needs to see the IngressClasses.

The role grants `get`, `list`, `watch` on the following resources:

- `storageclasses` (storage.k8s.io), plus `csidrivers`, `csinodes`, `volumeattachments`
- `volumesnapshotclasses` (snapshot.storage.k8s.io)
- `ingressclasses` (networking.k8s.io)
- `priorityclasses` (scheduling.k8s.io)
- `runtimeclasses` (node.k8s.io)
- `virtualmachineclasses`, `clustervirtualimages` (virtualization.deckhouse.io)
- `clusterlogdestinations` (deckhouse.io)
- `customresourcedefinitions` (apiextensions.k8s.io) — `get`, `list` only

Automatic binding

`user-authz-controller` keeps one ClusterRoleBinding to `d8:dict` for every subject that holds one of the following RoleBindings:

- a RoleBinding to a `d8:namespace:*` or `d8:project:*` role, including the RoleBindings that `multitenancy-manager` creates from a ProjectRoleBinding or a ClusterProjectRoleBinding;
- a RoleBinding that the module creates for an AuthorizationRule (the `User`, `PrivilegedUser`, `Editor`, and `Admin` levels).

The `system:multitenancy-manager:*` users, which `multitenancy-manager` acts as, get none. This means that project users can discover reference resources without any manual configuration. The binding is removed when the subject has no such RoleBinding left. You do not create it yourself.

> **i** The `d8:dict` role is independent of the `multitenancy-manager` cluster-wide resource access management mechanism: it only gives **read** access to reference resources so users can discover them. The mechanism controls *which resource values* a project may actually reference when creating objects.

Subsystems of the role-based model

Each Deckhouse Platform module belongs to a specific subsystem. For each subsystem, there is a set of roles with different levels of access. Roles are updated automatically when the module is enabled or disabled.

The subsystems of the platform are `iam`, `security`, `cluster`, `delivery`, `network`, `storage`, `observability` and `managed-services`.

For example, for the `network` subsystem, there are the following subsystem roles that can be used in `ClusterRoleBinding`:

- `d8:subsystem:network:viewer`;
- `d8:subsystem:network:manager`;
- `d8:subsystem:network:superadmin`.

The scope of a role depends on which subsystem it belongs to:

- The scope of the `d8:system:*` roles is the namespaces of all enabled Deckhouse Platform modules, that is, the namespaces that their capabilities name in the `rbac.deckhouse.io/namespace` label.
- The scope of subsystem roles includes the namespaces in which the subsystem's modules operate (see the subsystem composition table), as well as all cluster-wide objects of the subsystem's modules.

Role-based model subsystems composition table.

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
all	All modules	All namespaces
iam	• multitenancy-manager • namespace-configurator • user-authn • user-authz	• d8-multitenancy-manager • d8-system • d8-user-authn

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
security	• admission-policy-engine • cert-manager • neuvector • operator-trivy • runtime-audit-engine • secret-copier • secrets-store-integration	• d8-admission-policy-engine • d8-cert-manager • d8-operator-trivy • d8-runtime-audit-engine
cluster	• chrony • cloud-provider-aws • cloud-provider-azure • cloud-provider-dynamix • cloud-provider-gcp • cloud-provider-huaweicloud • cloud-provider-openstack • cloud-provider-vcd • cloud-provider-vsphere • cloud-provider-yandex • cloud-provider-zvirt • commander-agent • console • control-plane-manager • csi-vsphere • dashboard • deckhouse • deckhouse-tools • descheduler • documentation • node-manager • pod-reloader • registry • registry-packages-proxy • terraform-manager • vertical-pod-autoscaler • virtualization	• d8-chrony • d8-cloud-instance-manager • d8-cloud-provider-aws • d8-cloud-provider-azure • d8-cloud-provider-dynamix • d8-cloud-provider-gcp • d8-cloud-provider-huaweicloud • d8-cloud-provider-openstack • d8-cloud-provider-vcd • d8-cloud-provider-vsphere • d8-cloud-provider-yandex • d8-cloud-provider-zvirt • d8-csi-vsphere • d8-descheduler • d8-system • kube-system
delivery	• delivery-kit • operator-helm	• d8-operator-helm
network	• cilium-hubble • cni-cilium • cni-flannel • cni-simple-bridge • ingress-nginx • istio • keepalived • kube-dns • kube-proxy • metallb • monitoring-ping • network-gateway • network-policy-engine • node-local-dns • openvpn • sdn • service-with-healthchecks • static-routing-manager	• d8-cni-cilium • d8-cni-flannel • d8-cni-simple-bridge • d8-ingress-nginx • d8-istio • d8-keepalived • d8-metallb • d8-monitoring • d8-network-gateway • d8-openvpn • d8-service-with-healthchecks • d8-static-routing-manager • d8-system • kube-system

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
storage	• csi-ceph • csi-hpe • csi-huawei • csi-netapp • csi-nfs • csi-s3 • csi-scsi-generic • csi-yadro-tatlin-unified • local-path-provisioner • sds-local-volume • sds-node-configurator • sds-replicated-volume • snapshot-controller • storage-volume-data-manager	
observability	• extended-monitoring • log-shipper • loki • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-kubernetes-control-plane • observability • okmeter • operator-prometheus • prometheus • prometheus-metrics-adapter • prometheus-pushgateway • upmeter	• d8-log-shipper • d8-monitoring • d8-okmeter • d8-operator-prometheus • d8-upmeter • kube-prometheus-pushgateway
managed-services	• managed-memcached • managed-postgres	

How the roles are built: aggregation and capabilities

No built-in role contains a list of permissions directly. Permissions are described in separate small cluster roles — **capabilities**. Each capability is responsible for one kind of action (for example, “view logs”, “manage quotas”, “connect to pods”) and contains concrete RBAC rules. A role (`d8:namespace:admin`, `d8:system:viewer`, etc.) is an empty ClusterRole with an aggregation rule (`aggregationRule`): Kubernetes automatically collects into it the rules from all capabilities with matching labels.

Membership of objects in the role model is defined by the `rbac.deckhouse.io/*` labels — for example, the label `rbac.deckhouse.io/aggregate-to-namespace-as: admin` includes a capability into the `d8:namespace:admin` role (and, thanks to the level hierarchy, into all levels above). The complete list of labels and annotations is in the reference below.

This design has two practical consequences:

- Deckhouse Platform modules extend the roles automatically: when a module is enabled, its capabilities are added to the corresponding built-in roles; when it is disabled, they are removed. The permission list of a role always matches the set of enabled modules.
- You can assemble your own roles from ready-made capabilities without writing RBAC rules by hand. How to do this is described in the FAQ.

The names of built-in roles and capabilities start with the `d8:` prefix. This namespace is reserved: you cannot create your own ClusterRole with a `d8:*` name — the only exception is the `d8:custom:*` prefix, which is dedicated to user-defined roles and capabilities. The labels `rbac.deckhouse.io/kind: role` and `rbac.deckhouse.io/kind: capability` are also reserved for built-in objects — use `custom-role` and `custom-capability` for your own.

Reference of role labels and annotations

Labels the role model uses on ClusterRole objects:

Label	Found on	Purpose
`rbac.deckhouse.io/kind`	All role model objects	Object type: `role` or `capability` — built-in (reserved), `custom-role` or `custom-capability` — user-defined. An object without this label is refused if it carries the aggregation label of a platform scope.
`rbac.deckhouse.io/scope`	Roles and capabilities	Scope: `namespace`, `project`, `subsystem`, `system`.
`rbac.deckhouse.io/subsystem`	Subsystem objects	The subsystem name (for example, `network`) — only with `scope: subsystem`.
`rbac.deckhouse.io/aggregate-to-<scope>-as: <level>`	Capabilities and lower-level roles	The aggregation rule: includes the object into the role of the given scope and level. `<scope>` is `system`, `namespace`, `project`, or a subsystem name; `<level>` is `viewer`, `user`, `manager`, `admin`, `superadmin`. These are exactly the labels referenced by `aggregationRule` selectors.
`rbac.deckhouse.io/capability`	Capabilities	The globally unique name of the capability (for example, `namespace-capability.kubernetes.view_logs`) — used to include the capability into a custom role selectively.
`rbac.deckhouse.io/use-role: <level>`	System and subsystem roles	Which namespace-role level the holder of this role automatically gets in the system namespaces of its subsystem. On the built-in roles: `viewer` → `viewer`, `manager` → `viewer`, `superadmin` → `superadmin`; a role that is not built into the platform gets `viewer` at most. The access is granted by automatically created RoleBinding objects (labelled `rbac.deckhouse.io/automated: "true"`).
`rbac.deckhouse.io/namespace: <namespace>`	Capabilities	The namespace of a module where a RoleBinding is automatically created for the holders of a system/subsystem role. Read only on the capabilities Deckhouse ships; on a custom capability it is ignored (how to give access to another namespace).
`rbac.deckhouse.io/delegatable: "true"`	The `d8:namespace:*`, `d8:project:*`, and user-defined roles	The role may be referenced by a RoleBinding inside project namespaces. Add this label to your custom roles that should be available in projects.
`rbac.deckhouse.io/deprecated: "true"`	Deprecated alias roles	The role is deprecated and will be removed; migrate the bindings to the new role.

Label	Found on	Purpose
`module`	Built-in objects	The name of the Deckhouse Platform module the object belongs to. Convenient for aggregation selectors (for example, all capabilities of one module).
`heritage: deckhouse`	Built-in objects	Marks a platform object. Must not be set on your own objects.

Annotations on ClusterRole objects:

Annotation	Added by	Purpose
`ru.meta.deckhouse.io/title`, `ru.meta.deckhouse.io/description`	Platform	The display title and description in Russian.
`en.meta.deckhouse.io/title`, `en.meta.deckhouse.io/description`	Platform	The display title and description in English.
`custom.meta.deckhouse.io/title`, `custom.meta.deckhouse.io/description`	Administrator	Overrides the display title/description; the only allowed modification of built-in roles.
`rbac.deckhouse.io/bindable-only-via`	Platform	The binding kinds the role is intended for (on project roles — `ProjectRoleBinding,ClusterProjectRoleBinding`). It is informational: the platform does not check it.
`rbac.deckhouse.io/disabled-for-direct-use-in-projects: "true"`	Administrator	Forbids granting the role in projects: existing bindings keep working, new ones cannot be created (details).
`rbac.deckhouse.io/deprecated-replaced-by`	Platform	On deprecated alias roles: the name of the role to migrate to.

Admin level restrictions and superadmin rights

The role model provides two administration levels:

- `admin` — the everyday administrator. Manages resources, quotas, and access within their scope, but cannot perform operations that would let them break out of that scope or disrupt platform components.
- `superadmin` — the “break-glass” administrator. Has all the rights of `admin` and can additionally perform dangerous operations. Grant this level deliberately and only to those who really need it.

Only the `superadmin` level allows **minting ServiceAccount tokens** (`kubectl create token`) **and making requests on behalf of a ServiceAccount** (`kubectl --as system:serviceaccount:...`). A ServiceAccount token is a ready-to-use identity: by obtaining the token of a platform component’s service account, one could gain its permissions far beyond the namespace. Therefore `admin` manages the `ServiceAccount` objects themselves (create, delete) but cannot mint tokens for them or act on their behalf.

`superadmin` also has some restrictions:

- Objects labeled `heritage: deckhouse` belong to the platform: nobody may create, modify or delete them, `superadmin` and cluster administrators included. Some platform components place such objects in user namespaces, for example the pods of a DexAuthenticator. `kubectl exec`, `kubectl attach`, `kubectl port-forward` and `kubectl debug` into such pods and `kubectl scale` of such Deployment, StatefulSet, ReplicaSet and ReplicationController objects are denied too. `kubectl rollout restart` of such objects is allowed. The users these checks do not apply to are listed in the FAQ of the `deckhouse` module.
- Resources created from a project template (the `heritage: multitenancy-manager` label) cannot be modified by **anyone**, including `superadmin` and a cluster administrator — they are managed exclusively by the project controller. To change such a resource, change the project template or the project itself.

- A role is assigned via RoleBinding and applies only in the namespace where it was granted: a `superadmin` of one namespace gets no special rights in another.

Built-in protection mechanisms of the role model

The role model is protected by a set of checks at the API server level. They require no configuration and prevent typical mistakes and privilege escalation attempts:

- **A scoped role cannot be granted cluster-wide.** A ClusterRoleBinding to the `d8:namespace:*` or `d8:project:*` roles (and their `d8:custom:*` variants) is rejected — otherwise a role designed for one namespace or project would apply in every namespace at once. Use a RoleBinding in the desired namespace, or ProjectRoleBinding/ClusterProjectRoleBinding for a project. A ClusterRoleBinding is only allowed for system and subsystem roles — those are cluster-scoped by nature.
- **A capability cannot be granted cluster-wide.** A ClusterRoleBinding to any capability (`d8:*-capability:*`, including custom ones) is rejected: a capability is a building block for roles, not a standalone role. Binding a capability via a RoleBinding in a single namespace is allowed.
- **Project management cannot be obtained through a custom role.** Creating a Role or ClusterRole that grants permissions to modify project management resources (`projects`, `projecttemplates`, `projectrolebindings`, `clusterprojectrolebindings`, `projectnamespaces`) is rejected. These permissions are granted only by the built-in `d8:project:*` roles. Without this protection, a namespace administrator could create a role with the right to create ProjectRoleBinding objects and grant themselves access to the whole project. Read-only roles on these resources are allowed.
- **A namespace or project role cannot include platform capabilities.** A custom role with `scope: namespace`, `scope: project`, or no scope may not select system or subsystem capabilities by the `rbac.deckhouse.io/capability` label, nor the `system` or a subsystem scope by the `rbac.deckhouse.io/aggregate-to-<scope>-as` label. No custom role may combine the capabilities of the `namespace`/`project` scopes with those of the `system`/`subsystem` scopes — this rules out a “super-role” combining access to applications and to the platform.
- **A custom capability carries rules and cannot have an `aggregationRule`.**
- **The name must match the scope**: `d8:custom:namespace:<name>`, `d8:custom:project:<name>`, `d8:custom:system:<name>`, `d8:custom:<subsystem>:<name>`, and `d8:custom:<scope>-capability:<name>` for a capability.
- **Only a custom capability may carry an `rbac.deckhouse.io/aggregate-to-<scope>-as` label of a platform scope.**
- **The `rbac.deckhouse.io/delegatable: "true"` label is accepted only on a role that aggregates only namespace and project capabilities.**
- **Custom roles cannot contain direct RBAC rules** — they may only aggregate capabilities. Permissions are described in separate custom capabilities, which keeps the contents of any role transparent. See the FAQ for details.

Display names of roles

Every built-in role and capability carries a localized title and description in annotations:

- `ru.meta.deckhouse.io/title`, `ru.meta.deckhouse.io/description` — in Russian;
- `en.meta.deckhouse.io/title`, `en.meta.deckhouse.io/description` — in English.

These annotations are used, for example, by the Deckhouse web UI when displaying the list of roles.

If the standard title does not fit (for example, you want to name roles in your company’s terms), add the `custom.meta.deckhouse.io/title` and `custom.meta.deckhouse.io/description` annotations to the role — the interface will show them instead of the standard ones. Example:

```
d8 k annotate clusterrole d8:namespace:admin custom.meta.deckhouse.io/title='Team administrator'
```

This is the only allowed modification of built-in roles: changing their rules, aggregation, or labels is rejected.

Deprecated role names

The previous role names of the granular model (`d8:manage:<subsystem>:<level>`, `d8:manage:all:<level>`, and `d8:use:role:<level>`) are deprecated and will be removed in the next release. For backward compatibility they are temporarily kept as alias roles. A binding to an alias aggregates the **new** role’s capabilities, except where the list of subsystems changed (see below). The rights of an alias are not identical to the pre-upgrade ones. `d8:use:role:admin` no longer grants ServiceAccount token minting or impersonation, which moved to `superadmin`. The `d8:manage:*` aliases are not `delegatable`. The `d8:use:role:*` aliases are, so a RoleBinding to them inside a project namespace keeps working, but a new ProjectRoleBinding or ClusterProjectRoleBinding to an old name is refused.

A ClusterRoleBinding to a `d8:manage:*` alias keeps the RoleBindings that the previous release created

for it in the namespaces of the modules of the old subsystem. For a `viewer` alias they are named `d8:use:viewer:binding:<ClusterRoleBinding name>` and refer to `d8:use:role:viewer`, for a `manager` alias `d8:use:user:binding:<ClusterRoleBinding name>` and `d8:use:role:user`. They bind the subjects of the ClusterRoleBinding and are kept in sync as before. The namespaces are those of the platform modules that were in the old subsystem and of the capabilities of modules that still carry the old labels, `rbac.deckhouse.io/kind: manage` together with the aggregation label of the old subsystem. For `d8:manage:all:<level>` they also include the namespaces that the subsystem roles of the modules with a subsystem of their own reach at its level, for example `d8-virtualization`. A binding to an alias therefore gets no RoleBinding in the namespace of a module that joined the subsystem later or of a module released on its own that already carries the new labels. A ClusterRole created by a user adds no namespace, whatever labels it carries. The RoleBindings that the previous release created for the binding stay while the binding refers to the old name, even in a namespace that is no longer among these, for example when a module released on its own moves to the new labels. A ClusterRoleBinding that is deleted and created again under the same name keeps them only in these namespaces. When you recreate the binding on a new role name, these RoleBindings are replaced by `d8:namespace:<level>:binding:<ClusterRoleBinding name>` ones in the namespaces of the modules of the new subsystem, with the namespace role of the new role (`admin` for a `manager` role). The RoleBindings that the previous release created for a binding to a custom role of the previous scheme with `rbac.deckhouse.io/kind: manage`, such as `custom:manage:mycustom:manager`, are removed on the upgrade, because `user-authz-controller` keeps such RoleBindings for the roles of Deckhouse only. They come back as `d8:namespace:<level>:binding:<ClusterRoleBinding name>` once the role is recreated as a `d8:custom:*` role with `rbac.deckhouse.io/scope: system` or `subsystem` and `rbac.deckhouse.io/use-role` and the binding refers to the new role, but only in the namespaces of the modules that already carry the new labels. A module that still carries the old labels gets no such RoleBinding until it moves to the new ones. Until then, create RoleBinding objects in the namespace of that module for the subjects of the binding: one to `d8:namespace:<level>` and one to each ClusterRole of the module that the `D8UserAuthzLegacyUseCapability` alert names. None of these roles is an alias, so the RoleBinding objects do not hold the next release, and the webhook that checks RoleBinding objects in project namespaces does not check the namespace of a module. Delete them once the module carries the new labels and the controller creates its own RoleBinding there. A RoleBinding to `d8:use:role:<level>` gives these rights as well, but the `D8UserAuthzDeprecatedRBACv2RoleInUse` alert reports it, and it holds the release that removes the aliases.

Name mapping:

Deprecated name	New name
`d8:manage:all:<level>`	`d8:system:<level>`
`d8:manage:deckhouse:<level>`, `d8:manage:infrastructure:<level>`, `d8:manage:kubernetes:<level>`	`d8:subsystem:cluster:<level>`, which grants more. The modules that moved to other subsystems come with the roles of those subsystems
`d8:manage:networking:<level>`	`d8:subsystem:network:<level>`, which also grants the rights for `keepalived`
`d8:manage:security:<level>`	`d8:subsystem:security:<level>` together with `d8:subsystem:iam:<level>`, which grants the rights for `user-authz` and `user-authn` and more
`d8:manage:observability:<level>`	`d8:subsystem:observability:<level>`
`d8:manage:storage:<level>`	`d8:subsystem:storage:<level>`
`d8:use:role:<level>`	`d8:namespace:<level>`
`d8:use:role:<level>:kubernetes`	`d8:namespace:<level>` (full namespace role, not kubernetes-only)

The list of subsystems changed together with the role names. `deckhouse`, `infrastructure` and `kubernetes` were merged into `cluster`, which also took over modules from other subsystems, `networking` was renamed to `network`, and `user-authz`, `user-authn` and `multitenancy-manager` moved to `iam`.

The aliases of the `deckhouse`, `infrastructure` and `kubernetes` roles therefore do not aggregate the whole `cluster` subsystem. They select the capabilities of the platform modules that were in the old subsystem, wherever those modules are now. `d8:subsystem:cluster:<level>` grants more, so when you recreate such a binding, check which subsystem roles its subjects need. The alias of `d8:manage:security:<level>` also keeps the rights for `user-authz` and `user-authn`, which now come with `d8:subsystem:iam:<level>`. That role also grants the rights for `multitenancy-manager` and, at the `manager` level, the right to assign roles, which `d8:subsystem:security:<level>` does not grant. So a binding to `d8:manage:security:<level>` is replaced by two bindings with the same subjects, to `d8:subsystem:iam:<level>` and to `d8:subsystem:security:<level>`. The

`rbac.deckhouse.io/deprecated-replaced-by` annotation of the alias names only the second one. The alias of `networking` leaves out `keepalived`, which moved to `network` from `infrastructure`, so the new role grants more than this alias.

The `rbac.deckhouse.io/deprecated-replaced-by` annotation of an alias names its main replacement, and the description of the alias names the roles of the other subsystems its modules moved to. These roles cover whole subsystems. To grant the rights for some modules only, create a custom role that selects their system capabilities by the `rbac.deckhouse.io/capability` label, for example `system-capability.multitenancy-manager.view` (see How do I extend a role or create a new one?).

The aliases of the `deckhouse`, `infrastructure`, `kubernetes` and `networking` roles and `d8:manage:all:<level>` also select the labels of these old subsystems, such as `rbac.deckhouse.io/aggregate-to-deckhouse-as`. Modules released on their own and custom capabilities may still carry them. No role of the new model collects these labels, so such capabilities reach the holders of the old names only, and the `D8UserAuthzRetiredAggregationLabel` alert names them. A module released on its own that has moved to its new subsystem is no longer selected by the alias of its old one. `d8:manage:all:<level>` also selects `rbac.deckhouse.io/kind: manage` together with `rbac.deckhouse.io/aggregate-to-all-as` at its level, as the previous release did. Modules with a subsystem of their own carry this label on their subsystem roles, for example `d8:manage:virtualization:<level>`, until they move to the new labels.

The `d8:use:role:<level>` and `d8:use:role:<level>:kubernetes` aliases also select the use capabilities of the previous release, `rbac.deckhouse.io/kind: use` together with `rbac.deckhouse.io/aggregate-to-kubernetes-as` at the level of the alias or below. Modules released on their own and custom capabilities may still carry them, for example the `log-shipper` capabilities for PodLoggingConfig objects. `d8:use:role:<level>` also selects `rbac.deckhouse.io/kind: use` together with `rbac.deckhouse.io/aggregate-to-all-as` at its own level, as the previous release did. Modules with a subsystem of their own carry this label on their use roles, for example `d8:use:role:<level>:virtualization`. `d8:namespace:<level>` does not select them, so a RoleBinding to an old name keeps these rights, and a RoleBinding to the new name gets them only once the module or the capability carries `rbac.deckhouse.io/aggregate-to-namespace-as`. The `D8UserAuthzLegacyUseCapability` alert names the use capabilities with `rbac.deckhouse.io/aggregate-to-kubernetes-as` and the use roles with `rbac.deckhouse.io/aggregate-to-all-as`. The alert also names a ClusterRole with `rbac.deckhouse.io/kind: use` and the aggregation label of the system scope, of a built-in subsystem, or of `deckhouse`, `infrastructure` or `networking`. No role collects such a ClusterRole, so only a binding to the ClusterRole itself gives its rules. The `rbacv2-cluster-roles.deckhouse.io` webhook does not let a user create such a ClusterRole, and lets a user change one left from the previous release only by removing the aggregation label from it.

Until a module carries the new labels, create a RoleBinding to the ClusterRole that `D8UserAuthzLegacyUseCapability` names, such as `d8:use:role:<level>:virtualization` or the use capability itself, for the subjects that need these rights, in the namespace where they have their RoleBinding to `d8:namespace:<level>`. Neither role is an alias, so these RoleBinding objects do not hold the next release. In a project namespace, the `cluster-objects-grants-validator.multitenancy.deckhouse.io` webhook admits such a RoleBinding only if the ClusterRole has the `rbac.deckhouse.io/delegatable: "true"` label, and the use capabilities of the previous release do not have it. Where the webhook refuses it, keep the RoleBinding on the old name. It keeps these rights, but it holds the release that removes the aliases, so update the module before that release and then move the RoleBinding to `d8:namespace:<level>`.

Within the modules it covers, a binding to an old name gets their capabilities as they are in this release. These differ from the previous release, because the modules added resources to them. For example, the `multitenancy-manager` capabilities include the ProjectRoleBinding, ClusterProjectRoleBinding and ProjectNamespace objects, and the `iam` and `security` managers can inspect the access of any subject.

> ⚠ The deprecated `d8:use:role:admin` role maps to `d8:namespace:admin` and, just like it, no longer grants the right to mint ServiceAccount tokens or impersonate ServiceAccounts — that now requires the `superadmin` level.

As long as bindings to the deprecated names remain in the cluster, the `D8UserAuthzDeprecatedRBACv2RoleInUse` (a binding to an alias role) and `D8UserAuthzDeprecatedRBACv2CapabilityInUse` (a binding to a deprecated name that no ClusterRole has, such as a capability of the previous release, so the binding grants nothing) alerts fire. The release that removes the aliases carries the `deprecatedRBACv2BindingsCount` release requirement: the upgrade stays pending, with the offending bindings listed in the `DeckhouseRelease` status, until every binding to an alias role is recreated on a new role name. A binding that grants nothing does not hold the release: removing the aliases changes nothing for it. An alias role here is a ClusterRole of the `user-authz` module with the `rbac.deckhouse.io/deprecated` label. A module released on its own may still ship roles and capabilities of the previous release under the deprecated names, such as `d8:manage:virtualization:<level>`, `d8:use:role:<level>:virtualization` or `d8:manage:permission:module:ingress-nginx:edit`. A binding to such a ClusterRole keeps working,

raises neither alert and does not hold the release. The alerts and the release requirement leave out the RoleBindings that `user-authz-controller` keeps for a ClusterRoleBinding to an alias, because these follow the ClusterRoleBinding.

A module that renames a role of its own keeps the old name the same way: a ClusterRole labelled `rbac.deckhouse.io/deprecated` in the module's `templates/rbacv2-compat/`, with the new name in the `rbac.deckhouse.io/deprecated-replaced-by` annotation. A binding to such an alias raises `D8UserAuthzDeprecatedRBACv2ModuleRoleInUse`, which names the new role, and does not hold the Deckhouse release: the module removes the alias in a release of its own. The annotation is required (an alias without it is ignored), and the alias may have any `d8:` name except the platform's own role families (`d8:system:`, `d8:subsystem:`, `d8:namespace:`, `d8:project:`, `d8:custom:`, `d8:dict`), the reserved prefixes `d8:binding:` and `d8:prb:`, and the previous platform names `d8:manage:` and `d8:use:`.

Migrate your existing RoleBinding and ClusterRoleBinding objects to the new role names. A ClusterRoleBinding named `user-authz:<rule>:additional-role:<role>` is rendered by `user-authz-controller` from `spec.additionalRoles` of the ClusterAuthorizationRule `<rule>`, and the controller creates it again when it is deleted. So change the role name in `spec.additionalRoles` of that rule, not the binding. A binding that a project template renders is changed in the ProjectTemplate the same way, because `multitenancy-manager` renders it again from the template. You can find bindings that still use the deprecated names, without the RoleBindings that `user-authz-controller` keeps for them, with the command:

```
d8 k get clusterrolebindings,rolebindings -A -o json \
  | jq -r '.items[] | select(.roleRef.name | test("^d8:(manage|use):")) |
select(.metadata.labels["rbac.deckhouse.io/automated"] != "true") | "\(.kind) \
(.metadata.namespace // "-") \(.metadata.name) -> \(.roleRef.name)"'
```

Basic role-based model

Features:

- Access is configured with the ClusterAuthorizationRule and AuthorizationRule custom resources, on top of Kubernetes RBAC;
- Manages access to scaling tools (the `allowScale` parameter of the ClusterAuthorizationRule or AuthorizationRule custom resource);
- Manages access to port forwarding (the `portForwarding` parameter of the ClusterAuthorizationRule or AuthorizationRule custom resource);
- Manages the list of allowed namespaces with a labelSelector (the `namespaceSelector` parameter of the ClusterAuthorizationRule custom resource);
- Namespace restrictions (multi-tenancy mode, `namespaceSelector`, `limitNamespaces`) are available in the BE, SE, SE+, EE, and CSE editions.

In addition to the RBAC, you can use a set of high-level roles in the module:

- `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
- `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
- `Editor` — the same as `PrivilegedUser` + can create, edit, and delete the objects that are usually required for application tasks;
- `Admin` — the same as `Editor` + can create and modify pods, create ServiceAccount tokens (`serviceaccounts/token`), manage Role, RoleBinding, and AuthorizationRule objects, and delete auxiliary objects (for example, ReplicaSet and the challenges and orders of the `acme.cert-manager.io` group);
- `ClusterEditor` — the same as `Editor` + can manage a limited set of cluster-wide objects used in application tasks (DaemonSet, StorageClass, CustomResourceDefinition, GatewayClass, and the objects that modules add at this level, for example, KeepalivedInstance). This role is best suited for cluster operators.
- `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage cluster-wide service objects (namespaces, quotas, NetworkPolicy, the proxy subresources of nodes, ClusterAuthorizationRule, ClusterRole, and ClusterRoleBinding). This role is best suited for cluster administrators. **Warning.** `ClusterAdmin` can create ClusterAuthorizationRule objects only up to the `ClusterAdmin` level, and ClusterRoleBinding objects only to roles whose permissions it already holds;
- `SuperAdmin` — can perform any actions with any objects (note that `namespaceSelector` and `limitNamespaces` restrictions remain valid).

> ⚠ Multi-tenancy mode restricts access to namespaced resources only. Subjects with the `ClusterAdmin` or `SuperAdmin` access level can still create cluster-wide bindings. The authorization webhook does not check control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`, so the namespace restrictions do not apply to them.

If a ClusterAuthorizationRule custom resource contains the `namespaceSelector` field, the deprecated `limitNamespaces` and `allowAccessToSystemNamespaces` fields are not taken into consideration.

In multi-tenancy mode, if the authorization webhook becomes unavailable, requests are **denied**, not allowed: the webhook runs before RBAC and is configured with `failurePolicy: Deny`, so a request it cannot answer does not reach RBAC. The exception is the identities listed in `matchConditions` of the AuthorizationConfiguration: control plane components, `kubernetes-api-proxy`, kubelets, the ServiceAccounts of the `kube-system` and `d8-*` namespaces, and anonymous kube-apiserver health probes (`/livez`, `/readyz`, `/healthz`) bypass the webhook, so the cluster keeps running and Deckhouse can restore the webhook.

In multi-tenancy mode on Kubernetes 1.34 or later, the API server also asks the authorization webhook whether the `user-authn` module has revoked the ID token of a user. The module revokes tokens on a `Lock`, `ResetPassword` or `Reset2FA` UserOperation, on a `Logout` UserOperation without the `logout` field, and when a User is deleted. The API server denies a revoked token within seconds with the `user-authn: the credentials of this user were revoked; sign in again` message. This check does not deny requests when the webhook is unavailable or the `user-authn` module is disabled; a revoked ID token then stays valid until it expires. The check does not apply to Basic Authentication, ServiceAccount tokens, client certificates and requests that are already running, such as `watch` or `exec`. The check is not configured when the `apiserver.authz.webhookURL` parameter of the `control-plane-manager` module is set.

Default access list for each role

Each next role inherits permissions from the previous roles. A role block shows only the permissions added by that role.

The list below includes:

- standard permissions from the basic role-based model (k8s permissions);
- permissions created by Deckhouse's built-in modules.

It does not include permissions for modules from source.

When enabled in a cluster, modules from source create permissions for the resources they provide. When a module from source is disabled, the permissions it created are removed.

To view the permissions created by source modules, use the command.

`verbs` aliases:

- read - `get`, `list`, `watch`
- read-write - `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, `update`
- write - `create`, `delete`, `deletecollection`, `patch`, `update`

Role `User`:

```
get,list:
    - registry.deckhouse.io/registryimages
read:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - apps/deployments
    - apps/replicasets
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - cert-manager.io/certificaterequests
    - cert-manager.io/certificates
    - cert-manager.io/clusterissuers
    - cert-manager.io/issuers
    - cilium.io/ciliumclusterwidenetworkpolicies
    - cilium.io/ciliumnetworkpolicies
    - config.gatekeeper.sh/configs
```

```
    - configmaps
    - connection.gatekeeper.sh/connections
    - constraints.gatekeeper.sh/*
    - deckhouse.io/applicationpackages
    - deckhouse.io/applicationpackageversions
    - deckhouse.io/applications
    - deckhouse.io/awsinstanceclasses
    - deckhouse.io/azureinstanceclasses
    - deckhouse.io/baremetalimages
    - deckhouse.io/baremetalinstanceclasses
    - deckhouse.io/baremetalinstances
    - deckhouse.io/baremetalipxefirmwares
    - deckhouse.io/baremetalramdiskimages
    - deckhouse.io/clusterprojectrolebindings
    - deckhouse.io/deschedulers
    - deckhouse.io/dexauthenticators
    - deckhouse.io/dexclients
    - deckhouse.io/dynamixinstanceclasses
    - deckhouse.io/gcpinstanceclasses
    - deckhouse.io/huaweicloudinstanceclasses
    - deckhouse.io/hubblemonitoringconfigs
    - deckhouse.io/instances
    - deckhouse.io/keepalivedinstances
    - deckhouse.io/nodeextensionrequests
    - deckhouse.io/nodegroups
    - deckhouse.io/nodeoperations
    - deckhouse.io/openstackinstanceclasses
    - deckhouse.io/operationpolicies
    - deckhouse.io/projectnamespaces
    - deckhouse.io/projectrolebindings
    - deckhouse.io/projecttemplates
    - deckhouse.io/registryoperations
    - deckhouse.io/securitypolicies
    - deckhouse.io/securitypolicyexceptions
    - deckhouse.io/vcdaffinityrules
    - deckhouse.io/vcdinstanceclasses
    - deckhouse.io/vsphereinstanceclasses
    - deckhouse.io/yandexinstanceclasses
    - deckhouse.io/zvirtinstanceclasses
    - discovery.k8s.io/endpointslices
    - endpoints
    - events
    - events.k8s.io/events
    - expansion.gatekeeper.sh/expansiontemplate
    - extensions.istio.io/wasmplugins
    - extensions/daemonsets
    - extensions/deployments
    - extensions/ingresses
    - extensions/replicasets
    - extensions/replicationcontrollers
    - externaldata.gatekeeper.sh/providers
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gatewayclasses
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - infrastructure.cluster.x-k8s.io/dynamixclusters
    - infrastructure.cluster.x-k8s.io/dynamixmachines
    - infrastructure.cluster.x-k8s.io/dynamixmachinetemplates
    - infrastructure.cluster.x-k8s.io/huaweicloudclusters
    - infrastructure.cluster.x-k8s.io/huaweicloudmachines
    - infrastructure.cluster.x-k8s.io/huaweicloudmachinetemplates
    - infrastructure.cluster.x-k8s.io/vcdclusters
    - infrastructure.cluster.x-k8s.io/vcdclustertemplates
    - infrastructure.cluster.x-k8s.io/vcdmachines
    - infrastructure.cluster.x-k8s.io/vcdmachinetemplates
    - infrastructure.cluster.x-k8s.io/zvirtclusters
    - infrastructure.cluster.x-k8s.io/zvirtmachines
    - infrastructure.cluster.x-k8s.io/zvirtmachinetemplates
    - ipam.metal3.io/ipaddresses
    - ipam.metal3.io/ipclaims
    - ipam.metal3.io/ippools
    - ironic.metal3.io/ironics
    - limitranges
    - metal3.io/baremetalhosts
    - metal3.io/bmceventsubscriptions
    - metal3.io/dataimages
```

```
    - metal3.io/firmwareschemas
    - metal3.io/hardwaredata
    - metal3.io/hostclaims
    - metal3.io/hostdeploypolicies
    - metal3.io/hostfirmwarecomponents
    - metal3.io/hostfirmwaresettings
    - metal3.io/hostupdatepolicies
    - metal3.io/preprovisioningimages
    - metrics.k8s.io/nodes
    - metrics.k8s.io/pods
    - multitenancy.deckhouse.io/availableclusterresources
    - mutations.gatekeeper.sh/assign
    - mutations.gatekeeper.sh/assignimage
    - mutations.gatekeeper.sh/assignmetadata
    - mutations.gatekeeper.sh/modifyset
    - namespaces
    - network.deckhouse.io/clusternetworkpeers
    - network.deckhouse.io/egressgatewaypolicies
    - network.deckhouse.io/egressgateways
    - network.deckhouse.io/metalloadbalancerbgppeers
    - network.deckhouse.io/metalloadbalancerclasses
    - network.deckhouse.io/metalloadbalancerconfigurations
    - network.deckhouse.io/metalloadbalancerpools
    - network.deckhouse.io/networkpeerings
    - network.deckhouse.io/servicewithhealthchecks
    - network.deckhouse.io/waypointinstances
    - networking.istio.io/destinationrules
    - networking.istio.io/envoyfilters
    - networking.istio.io/gateways
    - networking.istio.io/serviceentries
    - networking.istio.io/sidecars
    - networking.istio.io/virtualservices
    - networking.istio.io/workloadentries
    - networking.istio.io/workloadgroups
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - node-manager.deckhouse.io/fencingfailednodestates
    - node-manager.deckhouse.io/fencingslaprofiles
    - nodes
    - persistentvolumeclaims
    - persistentvolumes
    - pods
    - pods/log
    - policy/poddisruptionbudgets
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - replicationcontrollers
    - resourcequotas
    - security.istio.io/authorizationpolicies
    - security.istio.io/peerauthentications
    - security.istio.io/requestauthentications
    - serviceaccounts
    - services
    - status.gatekeeper.sh/configpodstatuses
    - status.gatekeeper.sh/connectionpodstatuses
    - status.gatekeeper.sh/constraintpodstatuses
    - status.gatekeeper.sh/constrainttemplatepodstatuses
    - status.gatekeeper.sh/expansiontemplatepodstatuses
    - status.gatekeeper.sh/mutatorpodstatuses
    - status.gatekeeper.sh/providerpodstatuses
    - storage.k8s.io/storageclasses
    - syncset.gatekeeper.sh/syncsets
    - telemetry.istio.io/telemetries
    - templates.gatekeeper.sh/constrainttemplates
```

Role `PrivilegedUser` (includes all rules from the role `User`):

```
create:
    - pods/eviction
create,get:
    - pods/attach
    - pods/exec
delete,deletecollection:
    - pods
read:
    - secrets
```

Role `Editor` (includes all rules from the role `User`, `PrivilegedUser`):

```
get,patch:
    - pods/resize
write:
    - apps/deployments
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - cert-manager.io/certificates
    - configmaps
    - deckhouse.io/dexauthenticators
    - deckhouse.io/dexclients
    - discovery.k8s.io/endpointslices
    - endpoints
    - extensions/deployments
    - extensions/ingresses
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - network.deckhouse.io/servicewithhealthchecks
    - network.deckhouse.io/waypointinstances
    - networking.istio.io/destinationrules
    - networking.istio.io/gateways
    - networking.istio.io/serviceentries
    - networking.istio.io/sidecars
    - networking.istio.io/virtualservices
    - networking.istio.io/workloadentries
    - networking.istio.io/workloadgroups
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - persistentvolumeclaims
    - policy/poddisruptionbudgets
    - secrets
    - security.istio.io/authorizationpolicies
    - security.istio.io/peerauthentications
    - security.istio.io/requestauthentications
    - serviceaccounts
    - services
```

Role `Admin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create:
    - serviceaccounts/token
create,patch,update:
    - pods
delete,deletecollection:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - apps/replicasets
    - cert-manager.io/certificaterequests
    - extensions/replicasets
read-write:
    - deckhouse.io/authorizationrules
write:
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - cert-manager.io/issuers
    - deckhouse.io/applications
    - extensions.istio.io/wasmplugins
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - telemetry.istio.io/telemetries
```

Role `ClusterEditor` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create,delete,patch:
    - deckhouse.io/nodeoperations
delete,deletecollection:
    - acme.cert-manager.io/challenges
    - acme.cert-manager.io/orders
    - cert-manager.io/certificaterequests
get,list:
    - templates.internal.deckhouse.io/nodeconfigtemplates
patch,update:
    - nodes
read:
    - deckhouse.io/containerdintegritypolicies
    - deckhouse.io/ingressistiocontrollers
    - deckhouse.io/istiofederations
    - deckhouse.io/istiomulticlusters
    - multitenancy.deckhouse.io/grantableclusterresourcedefinitions
    - multitenancy.deckhouse.io/grantableclusterresourcereferences
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - sailoperator.io/istiocnis
    - sailoperator.io/istiorevisions
    - sailoperator.io/istiorevisiontags
    - sailoperator.io/istios
    - sailoperator.io/ztunnels
read-write:
    - deckhouse.io/nodegroupconfigurations
    - deckhouse.io/staticinstances
    - multitenancy.deckhouse.io/clusterresourcegrantpolicies
write:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - autoscaling.k8s.io/verticalpodautoscalercheckpoints
    - cert-manager.io/clusterissuers
    - cert-manager.io/issuers
    - deckhouse.io/applications
    - deckhouse.io/hubblemonitoringconfigs
    - deckhouse.io/instances
    - deckhouse.io/keepalivedinstances
    - deckhouse.io/nodeextensionrequests
    - deckhouse.io/nodegroups
    - extensions.istio.io/wasmplugins
    - extensions/daemonsets
    - gateway.networking.k8s.io/gatewayclasses
    - network.deckhouse.io/egressgatewaypolicies
    - network.deckhouse.io/egressgateways
    - storage.k8s.io/storageclasses
    - telemetry.istio.io/telemetries
```

Role `ClusterAdmin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`):

```
create:
    - authorization.deckhouse.io/bulksubjectaccessreviews
    - authorization.deckhouse.io/roleaccessreports
    - authorization.deckhouse.io/subjectaccessreports
    - deckhouse.io/dexauthenticators/allow-access-to-kubernetes
    - deckhouse.io/dexclients/allow-access-to-kubernetes
create,delete,patch,update:
    - deckhouse.io/registryoperations
get,list,patch,update,watch:
    - control-plane.deckhouse.io/controlplanenodes
patch,update:
    - deckhouse.io/vcdaffinityrules
    - infrastructure.cluster.x-k8s.io/dynamixclusters
    - infrastructure.cluster.x-k8s.io/dynamixmachines
    - infrastructure.cluster.x-k8s.io/dynamixmachinetemplates
    - infrastructure.cluster.x-k8s.io/huaweicloudclusters
    - infrastructure.cluster.x-k8s.io/huaweicloudmachines
    - infrastructure.cluster.x-k8s.io/huaweicloudmachinetemplates
    - infrastructure.cluster.x-k8s.io/vcdclusters
    - infrastructure.cluster.x-k8s.io/vcdclustertemplates
    - infrastructure.cluster.x-k8s.io/vcdmachines
    - infrastructure.cluster.x-k8s.io/vcdmachinetemplates
    - infrastructure.cluster.x-k8s.io/zvirtclusters
    - infrastructure.cluster.x-k8s.io/zvirtmachines
    - infrastructure.cluster.x-k8s.io/zvirtmachinetemplates
    - node-manager.deckhouse.io/fencingfailednodestates
proxy:
    - nodes
read:
```

```
    - control-plane.deckhouse.io/controlplaneoperations
    - deckhouse.io/dexproviders/sensitive
    - deckhouse.io/effectivelicenses
    - deckhouse.io/modulesettingsdefinitions
    - deckhouse.io/nodestaticpodrequests
    - deckhouse.io/projectquotastatuses
    - deckhouse.io/registryconfigs
    - deckhouse.io/registryconfigs/sensitive
    - deckhouse.io/registrymirrors/sensitive
    - deckhouse.io/registrynodes
    - deckhouse.io/registrystorages
    - deckhouse.io/registryupstreams/sensitive
    - deckhouse.io/useroperations/sensitive
    - deckhouse.io/users/sensitive
    - nfd.k8s-sigs.io/nodefeaturegroups
    - nfd.k8s-sigs.io/nodefeaturerules
    - nfd.k8s-sigs.io/nodefeatures
read-write:
    - deckhouse.io/clusterauthorizationrules
    - deckhouse.io/clusterlicenses
    - deckhouse.io/deckhousereleases
    - deckhouse.io/dexproviderchecks
    - deckhouse.io/dexproviders
    - deckhouse.io/groups
    - deckhouse.io/moduleconfigs
    - deckhouse.io/modulepulloverrides
    - deckhouse.io/modulereleases
    - deckhouse.io/modules
    - deckhouse.io/modulesources
    - deckhouse.io/moduleupdatepolicies
    - deckhouse.io/nodeusers
    - deckhouse.io/packagerepositories
    - deckhouse.io/packagerepositoryoperations
    - deckhouse.io/registrymirrors
    - deckhouse.io/registryupstreams
    - deckhouse.io/sshcredentials
    - deckhouse.io/tokenexchangeproviders
    - deckhouse.io/useraccounts
    - deckhouse.io/useroperations
    - deckhouse.io/users
    - nodes/configz
    - nodes/healthz
    - nodes/log
    - nodes/metrics
    - nodes/pods
    - nodes/proxy
    - nodes/stats
update:
    - deckhouse.io/projects/quota
    - namespaces/finalize
write:
    - cilium.io/ciliumclusterwidenetworkpolicies
    - cilium.io/ciliumnetworkpolicies
    - config.gatekeeper.sh/configs
    - connection.gatekeeper.sh/connections
    - constraints.gatekeeper.sh/*
    - deckhouse.io/applicationpackages
    - deckhouse.io/applicationpackageversions
    - deckhouse.io/awsinstanceclasses
    - deckhouse.io/azureinstanceclasses
    - deckhouse.io/baremetalimages
    - deckhouse.io/baremetalinstanceclasses
    - deckhouse.io/baremetalinstances
    - deckhouse.io/baremetalipxefirmwares
    - deckhouse.io/baremetalramdiskimages
    - deckhouse.io/clusterprojectrolebindings
    - deckhouse.io/containerdintegritypolicies
    - deckhouse.io/deschedulers
    - deckhouse.io/dynamixinstanceclasses
    - deckhouse.io/gcpinstanceclasses
    - deckhouse.io/huaweicloudinstanceclasses
    - deckhouse.io/ingressistiocontrollers
    - deckhouse.io/istiofederations
    - deckhouse.io/istiomulticlusters
    - deckhouse.io/openstackinstanceclasses
    - deckhouse.io/operationpolicies
    - deckhouse.io/projectnamespaces
    - deckhouse.io/projectrolebindings
    - deckhouse.io/projects
    - deckhouse.io/projecttemplates
    - deckhouse.io/securitypolicies
    - deckhouse.io/securitypolicyexceptions
```

```
    - deckhouse.io/vcdinstanceclasses
    - deckhouse.io/vsphereinstanceclasses
    - deckhouse.io/yandexinstanceclasses
    - deckhouse.io/zvirtinstanceclasses
    - expansion.gatekeeper.sh/expansiontemplate
    - externaldata.gatekeeper.sh/providers
    - limitranges
    - mutations.gatekeeper.sh/assign
    - mutations.gatekeeper.sh/assignimage
    - mutations.gatekeeper.sh/assignmetadata
    - mutations.gatekeeper.sh/modifyset
    - namespaces
    - network.deckhouse.io/metalloadbalancerbgppeers
    - network.deckhouse.io/metalloadbalancerclasses
    - network.deckhouse.io/metalloadbalancerconfigurations
    - network.deckhouse.io/metalloadbalancerpools
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - resourcequotas
    - sailoperator.io/istiocnis
    - sailoperator.io/istiorevisions
    - sailoperator.io/istiorevisiontags
    - sailoperator.io/istios
    - sailoperator.io/ztunnels
    - status.gatekeeper.sh/configpodstatuses
    - status.gatekeeper.sh/connectionpodstatuses
    - status.gatekeeper.sh/constraintpodstatuses
    - status.gatekeeper.sh/constrainttemplatepodstatuses
    - status.gatekeeper.sh/expansiontemplatepodstatuses
    - status.gatekeeper.sh/mutatorpodstatuses
    - status.gatekeeper.sh/providerpodstatuses
    - syncset.gatekeeper.sh/syncsets
    - templates.gatekeeper.sh/constrainttemplates
```

You can get additional list of access rules for module role from cluster (rules added by the administrator and non-default rules of other Deckhouse modules):

```
D8_ROLE_NAME=Editor
d8 k get clusterrole -o jsonpath="{range .items[?(@.metadata.annotations.user-authz\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Configuration

Available with limitations in: **Open/CE**

Available without limitations in: **Core, BE, SE, SE+, Ultimate/EE**

The module lifecycle stage: General Availability

Do not create Pods and ReplicaSets directly: these objects are secondary and should be created by controllers (Deployment, StatefulSet, Job, and others). The `User`, `PrivilegedUser`, `Editor`, and `ClusterEditor` access levels cannot create or modify Pods, and no access level below `SuperAdmin` can create or modify ReplicaSets.

In multi-tenancy mode, the `allowAccessToSystemNamespaces`, `namespaceSelector`, and `limitNamespaces` fields of ClusterAuthorizationRule are enforced by the authorization webhook. If the webhook is unavailable, the requests it has to answer are **denied**. The webhook does not check control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For details, see Basic role-based model.

Access rights are configured with the roles of the granular model in RoleBinding and ClusterRoleBinding objects, or with the custom resources of the basic model.

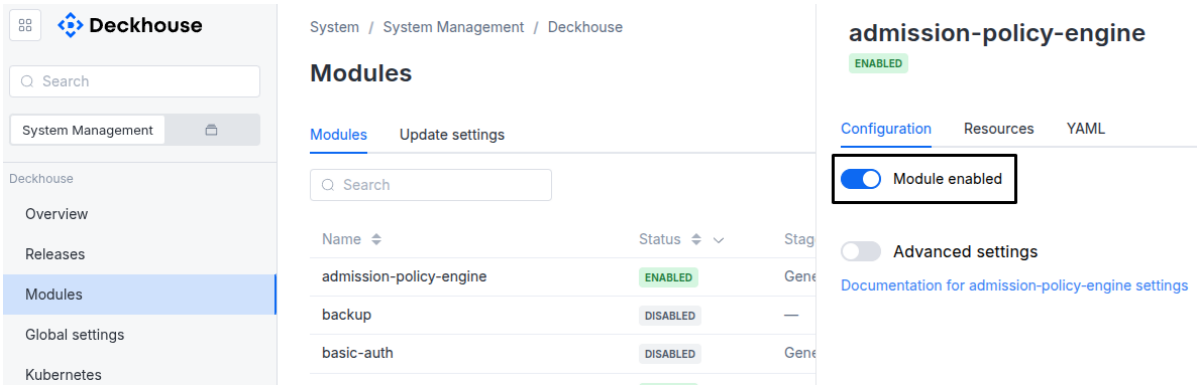
The module has 26 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authz` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable user-authz
```

- **Using ModuleConfig `user-authz`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `user-authz` (create it if necessary);

  Example of a manifest to enable module `user-authz`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authz
spec:
  enabled: true
```

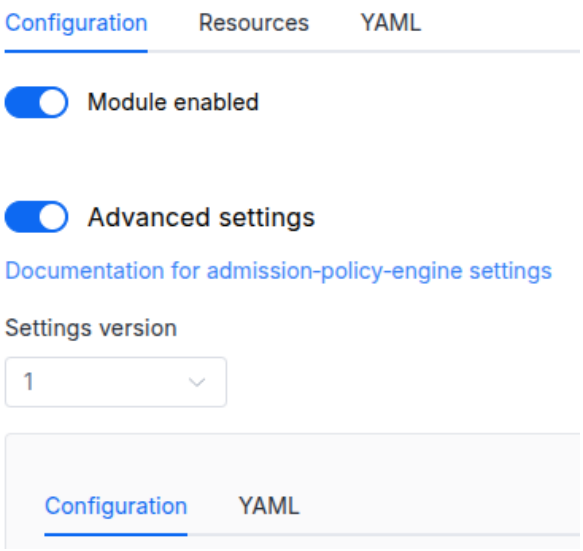
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authz` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `user-authz` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `user-authz`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `user-authz` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc user-authz
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `user-authz` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `user-authz`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: user-authz
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.controlPlaneConfigurator**
    object

    Parameters of the control-plane-manager module.

    - **settings.controlPlaneConfigurator.enabled**
      boolean

      Passes parameters for configuring `authz-webhook` to the `control-plane-manager` module (see the parameters of the control-plane-manager module).

If this parameter is disabled, the `control-plane-manager` module assumes that Webhook-based authorization is disabled by default. In this case (if no additional settings are provided), the `control-plane-manager` module will try to delete all references to the Webhook plugin from the manifest (even if you configure the manifest manually).

Default: `true`

Examples:

```
enabled: true
```

```
enabled: false
```

- **settings.enableMultiTenancy**
  boolean

  Available in editions: BE, SE, SE+, Ultimate/EE

  Enable namespace-based authorization (multi-tenancy mode).

  When enabled, the module deploys the authorization webhook on the master nodes and Permission Browser. They enforce the `namespaceSelector`, `limitNamespaces`, and `allowAccessToSystemNamespaces` fields of ClusterAuthorizationRule and serve WhoCan, SubjectAccessReport, and the filtered list of namespaces. The `kube-apiserver` settings are applied automatically by the control-plane-manager module, see Configuring kube-apiserver for multi-tenancy mode.

  Default: `false`

  Examples:

```
enableMultiTenancy: true
```

```
enableMultiTenancy: false
```

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom Resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

AuthorizationRule

Scope: Namespaced
Version: v1alpha1

Manages RBAC and authorization settings within a particular namespace.

> The AuthorizationRule resource belongs to the basic role-based model. Its permissions add up with those of the granular model.

- **status**
  object

Observed state of the rule, maintained by user-authz-controller.

- **status.bindings**
  integer

  Number of bindings rendered for the rule.

- **status.conditions**
  array of objects

  Conditions of the rule. `Ready=True` means every binding of the rule is applied; `Ready=False` with reason `InvalidSpec` means the rule cannot be rendered and no binding was changed, with reason `ApplyError` that the API server rejected a write (the message names the binding and the error).

  - **status.conditions.lastTransitionTime**
    string

    Time of the last change of the condition status.

  - **status.conditions.message**
    string

    Details of the status.

  - **status.conditions.observedGeneration**
    integer

    Generation of the rule that the condition refers to.

  - **status.conditions.reason**
    string

    Reason of the status, `BindingsApplied`, `InvalidSpec` or `ApplyError`.

  - **status.conditions.status**
    string

    Required value

    Condition status.

    Allowed values: `True`, `False`, `Unknown`

  - **status.conditions.type**
    string

    Required value

    Condition type. The controller sets `Ready`.

- **status.observedGeneration**
  integer

  Generation of the rule the status refers to.

- **spec**
  object

  Required value

  - **spec.accessLevel**
    string

    Access level:

    - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
    - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
    - `Editor` — the same as `PrivilegedUser` + can create, edit, and delete the objects that are usually required for application tasks;
    - `Admin` — the same as `Editor` + can create and modify pods, create ServiceAccount tokens (`serviceaccounts/token`), manage `Role`, `RoleBinding`, and `AuthorizationRule` objects, and delete auxiliary objects (for example, `ReplicaSet` and the challenges and orders of the `acme.cert-manager.io` group).

    Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`

Example:

```
accessLevel: PrivilegedUser
```

- **spec.allowScale**
  boolean

  Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

  Default: `false`

- **spec.portForwarding**
  boolean

  Allow/disallow the user to do `port-forwarding`.

  Default: `false`

- **spec.subjects**
  array of objects

  Required value

  Users, groups, and ServiceAccounts to grant the access level to.

  Kubernetes API reference

  Pay attention to the following nuances if this module is used together with the user-authn module:

  - Use the user's `email` as the username to grant privileges to the specific user;
  - When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
  - **spec.subjects.kind**
    string

    Required value

    Type of user identification resource.

    Allowed values: `User`, `Group`, `ServiceAccount`

    Example:

    ```
    kind: Group
    ```

  - **spec.subjects.name**
    string

    Required value

    Resource name.

    Example:

    ```
    name: some-group-name
    ```

  - **spec.subjects.namespace**
    string

    ServiceAccount namespace.

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

ClusterAuthorizationRule

Scope: Cluster

- v1
- v1alpha1

This object manages RBAC and namespace-based authorization.

The settings determine which access level is assigned to the user and/or group.

> **i** The ClusterAuthorizationRule resource belongs to the basic role-based model. Its permissions add up with those of the granular model.

- **status**
  object

  Observed state of the rule, maintained by user-authz-controller.

  - **status.bindings**
    integer

    Number of bindings rendered for the rule.

  - **status.conditions**
    array of objects

    Conditions of the rule. `Ready=True` means every binding of the rule is applied; `Ready=False` with reason `InvalidSpec` means the rule cannot be rendered and no binding was changed, with reason `ApplyError` that the API server rejected a write (the message names the binding and the error).

    - **status.conditions.lastTransitionTime**
      string

      Time of the last change of the condition status.

    - **status.conditions.message**
      string

      Details of the status.

    - **status.conditions.observedGeneration**
      integer

      Generation of the rule that the condition refers to.

    - **status.conditions.reason**
      string

      Reason of the status, `BindingsApplied`, `InvalidSpec` or `ApplyError`.

    - **status.conditions.status**
      string

      Required value

      Condition status.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      Condition type. The controller sets `Ready`.

  - **status.observedGeneration**
    integer

    Generation of the rule the status refers to.

- **spec**
  object

  Required value

  - **spec.accessLevel**
    string

    Access level:

    - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
    - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and

delete pods (and thus, restart them);
- `Editor` — the same as `PrivilegedUser` + can create, edit, and delete the objects that are usually required for application tasks;
- `Admin` — the same as `Editor` + can create and modify pods, create ServiceAccount tokens (`serviceaccounts/token`), manage `Role`, `RoleBinding`, and `AuthorizationRule` objects, and delete auxiliary objects (for example, `ReplicaSet` and the challenges and orders of the `acme.cert-manager.io` group);
- `ClusterEditor` — the same as `Editor` + can manage a limited set of cluster-wide objects used in application tasks (`DaemonSet`, `StorageClass`, `CustomResourceDefinition`, `GatewayClass`, and the objects that modules add at this level, for example, `KeepalivedInstance`). This role is best suited for cluster operators.
- `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage cluster-wide service objects (namespaces, quotas, `NetworkPolicy`, the proxy subresources of nodes, `ClusterAuthorizationRule`, `ClusterRole`, and `ClusterRoleBinding`). This role is best suited for cluster administrators.

  Warning. `ClusterAdmin` can create ClusterAuthorizationRule objects only up to the `ClusterAdmin` level, and ClusterRoleBinding objects only to roles whose permissions it already holds.
- `SuperAdmin` — can perform any actions with any objects (note that `limitNamespaces` and `namespaceSelector` (see below) restrictions remain valid).

Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

Example:

```
accessLevel: PrivilegedUser
```

- **spec.additionalRoles**
  array of objects

  Additional roles to bind for subjects.

  Use the `accessLevel` parameter instead; this parameter is for exceptional cases.

  Example:

  ```
  additionalRoles:
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-admin
  ```

  - **spec.additionalRoles.apiGroup**
    string

    Required value

    API group of the role. Only `rbac.authorization.k8s.io` is supported.

    Example:

    ```
    apiGroup: rbac.authorization.k8s.io
    ```

  - **spec.additionalRoles.kind**
    string

    Required value

    Kind of the role.

    Allowed values: `ClusterRole`

    Example:

    ```
    kind: ClusterRole
    ```

  - **spec.additionalRoles.name**
    string

    Required value

Name of the role.

Example:

```
name: cluster-admin
```

- **spec.allowAccessToSystemNamespaces**
  Deprecated
  boolean

  Available in editions: BE, SE, SE+, Ultimate/EE

  Allow access to system namespaces (`kube-*`, `d8-*`, `default`).

  Requires the enableMultiTenancy parameter. While it is disabled, a rule with this field is refused, and a rule that already has it is applied without its namespace limits, in every namespace of the cluster.

  The namespace limits of the rule are enforced by the authorization webhook, so they do not apply to the subjects that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For them, the access level of the rule applies in every namespace.

  Deprecated. Use the namespaceSelector field instead.

  Default: `false`

- **spec.allowScale**
  boolean

  Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

  Default: `false`

- **spec.limitNamespaces**
  Deprecated
  array of strings

  Available in editions: BE, SE, SE+, Ultimate/EE

  List of regex-patterns that define namespaces accessible by the user.

  Each pattern must match the whole namespace name: `team` does not match `team-2`. A pattern that is not a valid RE2 regular expression is refused when the rule is written. If the check misses it, the pattern is ignored, the rule opens fewer namespaces, and the `D8UserAuthzWebhookRulesQuarantined` alert fires.

  The decision making process:

  - If the list is defined, then only its constituents are accessible.
  - If the list is not defined, then all namespaces are accessible (except for the system ones, see the `allowAccessToSystemNamespaces` field).

  Requires the enableMultiTenancy parameter. While it is disabled, a rule with this field is refused, and a rule that already has it is applied without its namespace limits, in every namespace of the cluster.

  The namespace limits of the rule are enforced by the authorization webhook, so they do not apply to the subjects that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For them, the access level of the rule applies in every namespace.

  Deprecated. Use the namespaceSelector field instead.

  Example:

  ```
  limitNamespaces: production-.*
  ```

  - **Element of the array**
    string
- **spec.namespaceSelector**
  object

  Available in editions: BE, SE, SE+, Ultimate/EE

  Defines a set of namespaces accessible by the user, using the value of the

`namespaceSelector.labelSelector` parameter.

If the `namespaceSelector` parameter is specified, the values of the `limitNamespaces` and `allowAccessToSystemNamespaces` parameters are **ignored**. If the `namespaceSelector.matchAny` field is specified, then all namespaces (including system namespaces) will be accessible. Otherwise, only the namespaces with labels matching the `namespaceSelector.labelSelector` conditions will be accessible (including system namespaces).

If the `namespaceSelector` parameter is not specified, then the set of available namespaces is determined by the value of the `limitNamespaces` and `allowAccessToSystemNamespaces` parameters. If none of the parameters `namespaceSelector`, `limitNamespaces` and `allowAccessToSystemNamespaces` are specified, then all namespaces will be available, except for the system ones (`kube-*`, `d8-*`, `default`).

Requires the enableMultiTenancy parameter. While it is disabled, a rule with this field is refused, and a rule that already has it is applied without its namespace limits, in every namespace of the cluster.

The namespace limits of the rule are enforced by the authorization webhook, so they do not apply to the subjects that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For them, the access level of the rule applies in every namespace.

- **spec.namespaceSelector.labelSelector**
  object

  Defines the label selector-based filter of the namespaces.

  Cannot be used together with the `matchAny` parameter.

  If both `matchExpressions` and `matchLabels` parameters are set, their requirements are ANDed together — they must all be satisfied in order to match. If multiple `matchExpression` conditions are provided, they all must be satisfied in order to match.

  - **spec.namespaceSelector.labelSelector.matchExpressions**
    array of objects

    An array of set-based expressions.

    Example:

    ```
    matchExpressions:
    - key: tier
      operator: In
      values:
      - production
      - staging
    - key: tier
      operator: NotIn
      values:
      - production
    ```

    - **spec.namespaceSelector.labelSelector.matchExpressions.key**
      string

      A label name.

    - **spec.namespaceSelector.labelSelector.matchExpressions.operator**
      string

      A comparison operator.

      Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

    - **spec.namespaceSelector.labelSelector.matchExpressions.values**
      array of strings

      A label value.

      - **Element of the array**
        string

        Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Length: `1..63`

- **spec.namespaceSelector.labelSelector.matchLabels**
  object

  A set of labels a namespace must have to match.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.namespaceSelector.matchAny**
  boolean

  Unconditionally permits access to any namespace in the cluster, including system namespaces.

  Cannot be used together with the `labelSelector` parameter.

  Since `labelSelector` provides access to a limited number of the namespaces that match it, `matchAny` is needed in cases where access to all namespaces is to be granted.

  Allowed values: `true`

- **spec.portForwarding**
  boolean

  Allow/disallow the user to do `port-forwarding`.

  Default: `false`

- **spec.subjects**
  array of objects

  Required value

  Users, groups, and ServiceAccounts to grant the access level to.

  Kubernetes API reference

  Pay attention to the following nuances if this module is used together with the user-authn module:

  - Use the user's `email` as the username to grant privileges to the specific user;
  - When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
  - **spec.subjects.kind**
    string

    Required value

    Type of user identification resource.

    Allowed values: `User`, `Group`, `ServiceAccount`

    Example:

    ```
    kind: Group
    ```

  - **spec.subjects.name**
    string

    Required value

    Resource name.

    Example:

    ```
    name: some-group-name
    ```

  - **spec.subjects.namespace**
    string

ServiceAccount namespace. The namespace limits of the rule do not apply to the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`: the authorization webhook does not check them.

Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Length: `1..63`

Deprecated resource. Support for the resource might be removed in a later release.

This cluster-wide object manages RBAC and authorization.

> **i** The ClusterAuthorizationRule resource belongs to the basic role-based model. Its permissions add up with those of the granular model.

- **status**
  object

  Observed state of the rule, maintained by user-authz-controller.

  - **status.bindings**
    integer

    Number of bindings rendered for the rule.

  - **status.conditions**
    array of objects

    Conditions of the rule. `Ready=True` means every binding of the rule is applied; `Ready=False` with reason `InvalidSpec` means the rule cannot be rendered and no binding was changed, with reason `ApplyError` that the API server rejected a write (the message names the binding and the error).

    - **status.conditions.lastTransitionTime**
      string

      Time of the last change of the condition status.

    - **status.conditions.message**
      string

      Details of the status.

    - **status.conditions.observedGeneration**
      integer

      Generation of the rule that the condition refers to.

    - **status.conditions.reason**
      string

      Reason of the status, `BindingsApplied`, `InvalidSpec` or `ApplyError`.

    - **status.conditions.status**
      string

      Required value

      Condition status.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      Condition type. The controller sets `Ready`.

  - **status.observedGeneration**
    integer

    Generation of the rule the status refers to.

- **spec**
  object

Required value

- **spec.accessLevel**
  string

  Access level:

  - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
  - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
  - `Editor` — the same as `PrivilegedUser` + can create, edit, and delete the objects that are usually required for application tasks;
  - `Admin` — the same as `Editor` + can create and modify pods, create ServiceAccount tokens (`serviceaccounts/token`), manage `Role`, `RoleBinding`, and `AuthorizationRule` objects, and delete auxiliary objects (for example, `ReplicaSet` and the challenges and orders of the `acme.cert-manager.io` group);
  - `ClusterEditor` — the same as `Editor` + can manage a limited set of cluster-wide objects used in application tasks (`DaemonSet`, `StorageClass`, `CustomResourceDefinition`, `GatewayClass`, and the objects that modules add at this level, for example, `KeepalivedInstance`). This role is best suited for cluster operators.
  - `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage cluster-wide service objects (namespaces, quotas, `NetworkPolicy`, the proxy subresources of nodes, `ClusterAuthorizationRule`, `ClusterRole`, and `ClusterRoleBinding`). This role is best suited for cluster administrators.

    Warning. `ClusterAdmin` can create ClusterAuthorizationRule objects only up to the `ClusterAdmin` level, and ClusterRoleBinding objects only to roles whose permissions it already holds.
  - `SuperAdmin` — can perform any actions with any objects (note that `limitNamespaces` (see below) restrictions remain valid).

  Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

  Example:

  ```
  accessLevel: PrivilegedUser
  ```

- **spec.additionalRoles**
  array of objects

  Additional roles to bind for subjects.

  Use the `accessLevel` parameter instead; this parameter is for exceptional cases.

  Example:

  ```
  additionalRoles:
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-admin
  ```

  - **spec.additionalRoles.apiGroup**
    string

    Required value

    API group of the role. Only `rbac.authorization.k8s.io` is supported.

    Example:

    ```
    apiGroup: rbac.authorization.k8s.io
    ```

  - **spec.additionalRoles.kind**
    string

    Required value

    Kind of the role.

    Allowed values: `ClusterRole`

Example:

```
kind: ClusterRole
```

- **spec.additionalRoles.name**
  string

  Required value

  Name of the role.

  Example:

  ```
  name: cluster-admin
  ```

- **spec.allowAccessToSystemNamespaces**
  Deprecated
  boolean

  Available in editions: BE, SE, SE+, Ultimate/EE

  Allow access to system namespaces (`kube-*`, `d8-*`, `default`).

  Requires the enableMultiTenancy parameter. While it is disabled, a rule with this field is refused, and a rule that already has it is applied without its namespace limits, in every namespace of the cluster.

  The namespace limits of the rule are enforced by the authorization webhook, so they do not apply to the subjects that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For them, the access level of the rule applies in every namespace.

  Deprecated. Use the namespaceSelector parameter (API version `v1` of the CR) instead.

  Default: `false`

- **spec.allowScale**
  boolean

  Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

  Default: `false`

- **spec.limitNamespaces**
  Deprecated
  array of strings

  Available in editions: BE, SE, SE+, Ultimate/EE

  List of regex-patterns that define namespaces accessible by the user.

  Each pattern must match the whole namespace name: `team` does not match `team-2`. A pattern that is not a valid RE2 regular expression is refused when the rule is written. If the check misses it, the pattern is ignored, the rule opens fewer namespaces, and the `D8UserAuthzWebhookRulesQuarantined` alert fires.

  The decision making process:

  - If the list is defined, then only its constituents are accessible.
  - If the list is not defined, then all namespaces are accessible (except for the system ones, see the `allowAccessToSystemNamespaces` field).

  Requires the enableMultiTenancy parameter. While it is disabled, a rule with this field is refused, and a rule that already has it is applied without its namespace limits, in every namespace of the cluster.

  The namespace limits of the rule are enforced by the authorization webhook, so they do not apply to the subjects that the webhook does not check: control plane components, kubelets, and the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`. For them, the access level of the rule applies in every namespace.

  Deprecated. Use the namespaceSelector parameter (API version `v1` of the CR) instead.

  Example:

```
limitNamespaces: production-.*
```

- **Element of the array**
  string

- **spec.portForwarding**
  boolean

  Allow/disallow the user to do `port-forwarding`.

  Default: `false`

- **spec.subjects**
  array of objects

  Required value

  Users, groups, and ServiceAccounts to grant the access level to.

  Kubernetes API reference

  Pay attention to the following nuances if this module is used together with the user-authn module:

  - Use the user's `email` as the username to grant privileges to the specific user;
  - When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
  - **spec.subjects.kind**
    string

    Required value

    Type of user identification resource.

    Allowed values: `User`, `Group`, `ServiceAccount`

    Example:

    ```
    kind: Group
    ```

  - **spec.subjects.name**
    string

    Required value

    Resource name.

    Example:

    ```
    name: some-group-name
    ```

  - **spec.subjects.namespace**
    string

    ServiceAccount namespace. The namespace limits of the rule do not apply to the ServiceAccounts of `kube-system` and of the namespaces whose names start with `d8-`: the authorization webhook does not check them.

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Usage

> The module lifecycle stage: General Availability

Example of assigning rights to a cluster administrator

> The example uses the granular role-based model.

To grant access to a cluster administrator, use the role `d8:system:manager` in ClusterRoleBinding.

Example of assigning rights to a cluster administrator (User `jane.doe@example.com`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:system:manager
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

The user gets access to cluster-wide resources, to the configuration of Deckhouse Platform modules, and to the namespaces of the enabled Deckhouse Platform modules. The user gets no access to the namespaces of user applications.

The user will be able to:

- View, modify, delete, and create Kubernetes resources and Deckhouse Platform modules;
- Modify module configurations (view, modify, delete, and create `moduleConfig` resources);
- Execute the following commands on pods and services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

Example of assigning rights to a network administrator

> The example uses the granular role-based model.

To grant a network administrator access to manage the network subsystem of the cluster, use the role `d8:subsystem:network:manager` in ClusterRoleBinding.

Example of assigning rights to a network administrator (User `jane.doe@example.com`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:subsystem:network:manager
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

The rights that the user will get will be limited to the following list of Deckhouse Platform module namespaces from the `network` subsystem (the actual list depends on the list of modules included in the cluster):

- `d8-cni-cilium`
- `d8-cni-flannel`
- `d8-cni-simple-bridge`
- `d8-ingress-nginx`
- `d8-istio`
- `d8-keepalived`
- `d8-metallb`
- `d8-monitoring`
- `d8-network-gateway`
- `d8-openvpn`
- `d8-service-with-healthchecks`
- `d8-static-routing-manager`
- `d8-system`
- `kube-system`

The user will be able to:

- View, modify, delete, and create *standard* Kubernetes resources in the module namespace from the `network` subsystem.

  Example of resources that the user will be able to manage (the list is not exhaustive):

  - `Certificate`
  - `CertificateRequest`
  - `ConfigMap`
  - `ControllerRevision`
  - `CronJob`
  - `DaemonSet`
  - `Deployment`
  - `Event`
  - `HorizontalPodAutoscaler`
  - `Ingress`
  - `Issuer`
  - `Job`
  - `Lease`
  - `LimitRange`
  - `NetworkPolicy`
  - `PersistentVolumeClaim`
  - `Pod`
  - `PodDisruptionBudget`
  - `ReplicaSet`
  - `ReplicationController`
  - `ResourceQuota`
  - `Role`
  - `RoleBinding`
  - `Secret`
  - `Service`
  - `ServiceAccount`
  - `StatefulSet`
  - `VerticalPodAutoscaler`
  - `VolumeSnapshot`

- View, modify, delete, and create the following resources in the modules namespace from the `network` subsystem:

  A list of resources that the user will be able to manage:

  - `EgressGateway`
  - `EgressGatewayPolicy`
  - `FlowSchema`
  - `IngressClass`
  - `IngressIstioController`
  - `IngressNginxController`
  - `IPRuleSet`
  - `IstioFederation`
  - `IstioMulticluster`
  - `KeepalivedInstance`
  - `RoutingTable`

- Modify the configuration of modules (view, change, delete, and create moduleConfig resources) from the `network` subsystem.

  List of modules that the user will be able to manage:

  - `cilium-hubble`
  - `cni-cilium`
  - `cni-flannel`
  - `cni-simple-bridge`
  - `ingress-nginx`
  - `istio`
  - `keepalived`
  - `kube-dns`
  - `kube-proxy`
  - `metallb`
  - `monitoring-ping`
  - `network-gateway`
  - `network-policy-engine`
  - `node-local-dns`
  - `openvpn`
  - `sdn`
  - `service-with-healthchecks`
  - `static-routing-manager`

- Execute the following commands with pods and services in the modules namespace from the `network` subsystem:

  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

Example of assigning administrative rights to a user within a namespace

> **i** The example uses the granular role-based model.

To assign rights to a user to manage application resources within a namespace (including some namespaced Deckhouse Platform module resources such as DexAuthenticator), use the role `d8:namespace:admin` in a RoleBinding in the corresponding namespace. Cluster-wide module configuration is not included.

Example of assigning rights to an application developer (User `app-developer`) in namespace `myapp`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:namespace:admin
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

In the `myapp` namespace, the user will be able to:

- View, modify, delete, and create Kubernetes resources. For example, the following resources (the list is not exhaustive):
  - `Certificate`
  - `CertificateRequest`
  - `ConfigMap`
  - `ControllerRevision`
  - `CronJob`
  - `DaemonSet`
  - `Deployment`
  - `Event`
  - `HorizontalPodAutoscaler`
  - `Ingress`
  - `Issuer`
  - `Job`
  - `Lease`
  - `LimitRange`
  - `NetworkPolicy`
  - `PersistentVolumeClaim`
  - `Pod`
  - `PodDisruptionBudget`
  - `ReplicaSet`
  - `ReplicationController`
  - `ResourceQuota`
  - `Role`
  - `RoleBinding`
  - `Secret`
  - `Service`
  - `ServiceAccount`
  - `StatefulSet`
  - `VerticalPodAutoscaler`
  - `VolumeSnapshot`
- View, edit, delete, and create the following Deckhouse Platform module resources:
  - `DexAuthenticator`
  - `DexClient`
  - `PodLoggingConfig`
- Execute the following commands for pods and services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

The user will **not** get the operations reserved for the `superadmin` level (see the section Admin level restrictions and superadmin rights): minting ServiceAccount tokens, making requests on behalf of ServiceAccounts, modifying platform system resources in this namespace, and connecting to system pods.

An example of assigning rights across all namespaces of a project

> **i** The example uses the granular role-based model and the multitenancy-manager module.

If the namespaces are grouped into a project, a role can be granted for the whole project at once — it will automatically apply in all of its namespaces, including those created later. To do that, use a ProjectRoleBinding in the main namespace of the project instead of RoleBinding:

```
apiVersion: deckhouse.io/v1alpha3
kind: ProjectRoleBinding
metadata:
  name: team-developers
  namespace: my-project
spec:
  subjects:
    - kind: Group
      name: developers
  roleRef:
    kind: ClusterRole
    name: d8:project:user
```

For more details on role bindings in projects, see the multitenancy-manager module documentation.

An example of a ClusterAuthorizationRule

> **i** The example uses the basic role-based model.

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
  - kind: User
    name: some@example.com
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci
  - kind: Group
    name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # This option is only available if the enableMultiTenancy parameter is set.
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: stage
        operator: In
        values:
        - test
        - review
      matchLabels:
        team: frontend
```

A `User` or `Group` subject is matched by name against the user identity in the token. A locally managed identity is indistinguishable from one asserted by an external authentication provider. Because of that, you can't create a User whose `spec.email` or a Group whose `spec.name` matches a subject of an existing ClusterAuthorizationRule or AuthorizationRule. This prevents any unnoticed attempt to grant the identity privileges.

If the match is intentional, for example, if the ClusterAuthorizationRule has been written in advance, set the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation on the User or Group. The annotation only acknowledges the name collision. It does not let a requester assign roles they cannot cover or that sit outside their can-assign range.

Deleting a ClusterAuthorizationRule uses the same can-assign check as creating or updating it. You can

remove a grant only if you could assign those roles. An update is checked against both the previous and the new grants, so lowering `SuperAdmin` to `User` still requires being able to assign `SuperAdmin`.

Write an email for the `User` subject in lowercase. It's recorded to the token in lowercase so a subject spelled `Admin@Example.com` won't match `admin@example.com`. Names of `Group` subjects are matched exactly, because group names are not lowercased anywhere.

This restriction is applied when a User or Group is created and when its `spec.email` or `spec.name` is changed. Deleting a User or Group whose name a rule still grants privileges to is allowed with a warning: the name stays granted, and recreating the object restores the privileges. The restriction doesn't prevent adding existing users or groups to a ClusterAuthorizationRule or AuthorizationRule. If a matching User or Group already exists, it receives the privileges immediately after it's added to the rule.

Managing permissions via CLI

> **i** The `d8 iam access` command uses the basic role-based model.

The `d8 iam access` command provides a convenient way to manage authorization rules without writing YAML manifests. `d8 iam access grant` creates ClusterAuthorizationRule and AuthorizationRule objects, and `d8 iam access revoke` deletes only the objects created by `grant`.

The command works with the basic role-based model, and with users and groups only. Grant roles of the granular model with RoleBinding and ClusterRoleBinding objects. Give access to a ServiceAccount with a manifest, see Creating a ServiceAccount for a machine and granting it access. The `d8 iam get user` and `d8 iam list users` commands count only ClusterAuthorizationRule and AuthorizationRule objects. To see all the access of a subject, use SubjectAccessReport.

Available access levels: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`.

Examples of granting permissions with the `d8 iam access` command:

Grant the Admin role to a user in the `dev` namespace:

```
d8 iam access grant user anton --access-level Admin -n dev
```

Grant the Admin role to a user in multiple namespaces:

```
d8 iam access grant user anton --access-level Admin -n dev -n stage
```

Grant a cluster-wide role to a user (excluding system namespaces):

```
d8 iam access grant user anton --access-level ClusterAdmin --scope cluster
```

Grant a cluster-wide role to a user in all namespaces (including system namespaces):

```
d8 iam access grant user anton --access-level ClusterAdmin --scope all-namespaces
```

Grant a role to group members by a namespace label selector:

```
d8 iam access grant group admins --access-level Editor --scope labels=team=platform,tier=prod
```

`--scope cluster` leaves out the system namespaces only in multi-tenancy mode; without it, the rule grants its access level in every namespace, system ones included. `--scope labels=...` and `--scope all-namespaces` need multi-tenancy mode: without it, the rule is refused. A rule created with `-n` is an AuthorizationRule and applies only in its namespace.

Grant a role to group members with additional capabilities:

```
d8 iam access grant group admins --access-level Editor -n dev --port-forwarding --allow-scale
```

Preview the manifest without applying it:

```
d8 iam access grant user anton --access-level Admin -n dev --dry-run -o yaml
```

Revoke a user’s permissions in a namespace:

```
d8 iam access revoke user anton --access-level Admin -n dev
```

Revoke a user’s cluster-wide permissions:

```
d8 iam access revoke user anton --access-level ClusterAdmin --scope cluster
```

The access level and the scope must match the original grant.

List all access rules in the cluster:

```
d8 iam list rules
```

Show detailed information about a specific rule. Name a ClusterAuthorizationRule as `CAR/<NAME>` and an AuthorizationRule as `AR/<NAMESPACE>/<NAME>`:

```
d8 iam get rule CAR/<NAME>
d8 iam get rule AR/<NAMESPACE>/<NAME>
```

Example of granting access to all namespaces

> **i** The example refers to the basic role-based model.

In multi-tenancy mode (`userAuthz.enableMultiTenancy`), namespace-based restrictions are configured using the ClusterAuthorizationRule resource fields.

Configuration options:

- **Allow access to all namespaces, including system namespaces** (`kube-*`, `d8-*`, `default`). To allow access to all namespaces, including system namespaces, set `spec.namespaceSelector.matchAny: true`. If the `namespaceSelector` parameter is specified, the values of the deprecated `limitNamespaces` and `allowAccessToSystemNamespaces` parameters are **ignored**. Access to the system namespaces with the `Admin` or `ClusterAdmin` level is equivalent to a cluster administrator: those namespaces hold the ServiceAccounts of the platform (see Access of system and subsystem roles to the namespaces of the modules).

  Example:

  ```
  apiVersion: deckhouse.io/v1
  kind: ClusterAuthorizationRule
  metadata:
    name: all-namespaces-including-system
  spec:
    subjects:
    - kind: User
      name: user@example.com
    accessLevel: Editor
    namespaceSelector:
      matchAny: true
  ```

- **Allow access to all namespaces except system namespaces**. To grant a user access to all namespaces except system namespaces, do not specify `namespaceSelector` or the deprecated `limitNamespaces` and `allowAccessToSystemNamespaces` parameters when creating a ClusterAuthorizationRule. For a list of system namespaces, see the CR field description.

> **i** The `SuperAdmin` access level **does not lift** the namespace restrictions specified in the `namespaceSelector` and `limitNamespaces` parameters. If you need to grant access to all namespaces, specify the scope explicitly, including via `namespaceSelector.matchAny`.

If several `ClusterAuthorizationRule` resources match the same subject, the allowed namespaces are **combined**; the effective `accessLevel` is the **most powerful** among all matching rules. For details, refer to the FAQ.

⚠ Namespace restrictions from ClusterAuthorizationRule are enforced by the authorization webhook. The API server calls it before RBAC with `failurePolicy: Deny`, so if the webhook is unavailable, the requests it has to answer are **denied**. The webhook does not check control plane components, kubelets, and the ServiceAccounts of the `kube-system` namespace and of the namespaces whose names start with `d8-`. Namespace restrictions do not apply to them. For details, see the module description.

Granular role-based model

In the granular model, bind `d8:namespace:*` roles with a RoleBinding in each namespace, or use a ProjectRoleBinding to cover every namespace of a project. `d8:system:*` / `d8:subsystem:*` roles do not cover namespaces for user workloads; they only apply to system namespaces (`d8-*`, `kube-*`) within the subsystem.

Creating a user

There are two types of users in Kubernetes:

- Service accounts managed by Kubernetes via the API;
- Regular users and groups managed by some external tool that the cluster administrator configures. There are many authentication mechanisms and, accordingly, many ways to create users. Currently, two authentication methods are supported:
  - Via the `user-authn` module. The module supports the following external providers and authentication protocols: GitHub, GitLab, Bitbucket Cloud, Atlassian Crowd, LDAP, OIDC, SAML, and local users (the User object). More details — in the documentation of the `user-authn` module.
  - Via the certificates.

When issuing the authentication certificate, you need to specify the name (`CN=<name>`), the required number of groups (`O=<group>`), and sign it using the root CA of the cluster. It is this mechanism that authenticates you in the cluster when, for example, you use kubectl on a master node.

Creating a ServiceAccount for a machine and granting it access

ℹ The example uses the basic role-based model.

You may need to create a ServiceAccount with access to the Kubernetes API when, for example, an application is deployed using a CI system.

1. Create a namespace for CI, for example `ci`, and a ServiceAccount in it:

```
d8 k create -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: ci
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: ci
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: ci
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

2. Grant it the necessary privileges (using the ClusterAuthorizationRule custom resource):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-deploy
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci
  accessLevel: Editor
EOF
```

   If multi-tenancy mode is enabled (the `enableMultiTenancy` parameter), limit the namespaces the ServiceAccount has access to with the `namespaceSelector` parameter.

   ⚠ Do not create ServiceAccounts for external systems in `kube-system` or in namespaces whose names start with `d8-`. The authorization webhook does not check the ServiceAccounts of these namespaces, so the namespace limits of a ClusterAuthorizationRule (`namespaceSelector`, `limitNamespaces`, `allowAccessToSystemNamespaces`) do not apply to them. Their access level applies in every namespace of the cluster, including system namespaces.

3. Set the variable values (they will be used later) by running the following commands (**insert your own values**):

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy.my-cluster
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config
```

4. Generate the `cluster` section in the kubectl configuration file:

   Use one of the following options to access the cluster API server:

   - If there is direct access to the API server:
     1. Get a Kubernetes cluster CA certificate:

```
d8 k get cm kube-root-ca.crt -o jsonpath='{ .data.ca\.crt }' > /tmp/ca.crt
```

2. Generate the `cluster` section (the API server's IP address is used for access):

```
d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
  --server=https://$(d8 k get endpointslices -l kubernetes.io/service-name=kubernetes
-o json | jq -rc '.items[0] | "\(.endpoints[0].addresses[0]):\
(.ports[0].port)"') \
  --certificate-authority=/tmp/ca.crt \
  --kubeconfig=$FILE_NAME
```

- If there is no direct access to the API server, use one of the following options:
  - enable access to the API-server over the Ingress controller (the apiserver.publishAPI parameter of the `control-plane-manager` module) and specify the addresses from which requests originate (the whitelistSourceRanges parameter);
  - specify addresses from which requests will originate in a separate Ingress controller (the acceptRequestsFrom parameter).
- If a non-public CA is used:

1. Get the CA certificate from the Secret of the `kubernetes-api` Ingress in the `kube-system` namespace. It is the certificate of the `api.%s` domain:

```
d8 k -n kube-system get secrets -o json \
  $(d8 k -n kube-system get ing kubernetes-api -o jsonpath="
{.spec.tls[0].secretName}") \
  | jq -rc '.data."ca.crt" // .data."tls.crt"' \
  | base64 -d > /tmp/ca.crt
```

2. Generate the `cluster` section (an external domain and a CA for access are used):

```
d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
  --server=https://$(d8 k -n kube-system get ing kubernetes-api -ojson | jq
'.spec.rules[].host' -r) \
  --certificate-authority=/tmp/ca.crt \
  --kubeconfig=$FILE_NAME
```

- If a public CA is used. Generate the `cluster` section (an external domain is used for access):

```
d8 k config set-cluster $CLUSTER_NAME \
  --server=https://$(d8 k -n kube-system get ing kubernetes-api -ojson | jq
'.spec.rules[].host' -r) \
  --kubeconfig=$FILE_NAME
```

- If the API server is published through Gateway API, the cluster has the `kubernetes-api` HTTPRoute in the `kube-system` namespace instead of the Ingress. Take the host from it instead of `.spec.rules[].host` in the commands above:

```
d8 k -n kube-system get httproute kubernetes-api -o jsonpath='{.spec.hostnames[0]}'
```

  Take the name of the Secret with the CA certificate from the `control-plane-manager` ListenerSet instead of `.spec.tls[0].secretName`:

```
d8 k -n kube-system get listenerset control-plane-manager -o
jsonpath='{.spec.listeners[?(@.name=="api")].tls.certificateRefs[0].name}'
```

5. Generate the `user` section using the token from the Secret's ServiceAccount in the kubectl configuration file:

```
d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n ci get secret gitlab-runner-deploy-token -o json |jq -r
'.data["token"]' | base64 -d) \
  --kubeconfig=$FILE_NAME
```

6. Generate the context in the kubectl configuration file:

```
d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME
```

7. Set the generated context as the default one in the kubectl configuration file:

```
d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

How to create a user using a client certificate

> **i** This method is recommended for system needs (authentication of kubelets, control plane components, etc.). If you need to create a “normal” user (e.g. with console access, `kubectl`, etc.), use kubeconfig generation.

To create a user with a client certificate, you can use either OpenSSL or the Kubernetes API (via the CertificateSigningRequest object).

> ⚠ Certificates issued by any of these methods cannot be revoked. If a certificate is compromised, you will need to remove all permissions for that user (this can be difficult if the user is added to any groups: you will also need to remove all relevant groups).

Creating a user using a certificate issued via OpenSSL

> ⚠ Consider security risks when using this method.
>
> The `ca.crt` and `ca.key` must not leave the master node: sign the CSR only on the master node.
>
> Signing CSRs outside the master node risks compromising the cluster root certificate.

The features of this method are:

- The client certificate must be signed on the master node to prevent the cluster certificate from being compromised.
- Access to the cluster CA key (`ca.key`) is required. Only the cluster administrator can sign certificates.

To create a user using a client certificate issued through OpenSSL, follow these steps:

1. Make sure you have access to a master node. The cluster CA files `ca.crt` and `ca.key` are in `/etc/kubernetes/pki/` on it.
2. Generate the user key:

```
openssl ecparam -name prime256v1 -genkey -out myuser.key
```

3. Create a CSR file and specify the username in it (`myuser`) and groups to which this user belongs (`mygroup1` and `mygroup2`):

```
openssl req -new -key myuser.key -out myuser.csr -subj
"/CN=myuser/O=mygroup1/O=mygroup2"
```

4. Upload the CSR created in the previous step (`myuser.csr` in this example) to the master node

and sign it with the cluster root certificate. Example command to sign the CSR on the master node (make sure that the paths to `myuser.csr`, `ca.crt` and `ca.key` are correct for your case):

```
openssl x509 -req -in myuser.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out myuser.crt -days 10
```

Now the certificate can be specified in the config file:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
    server: https://<cluster_host>:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: myuser
  name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Creating a user by issuing a certificate via the Kubernetes API

This is a more secure method because a special kubernetes API is used to sign the certificate.

The features of this method are:

- API-based certificate signing: CSRs are processed through the Kubernetes API without requiring access to the CA's private key (`ca.key`).
- Not only the cluster administrator can issue client certificates. The right to create CSRs and sign them can be assigned to a specific user.

To create a user using a client certificate issued through the Kubernetes API, follow these steps:

1. Generate the user key:

   ```
   openssl ecparam -name prime256v1 -genkey -out myuser.key
   ```

2. Create a CSR file and specify in it the username (`myuser`) and groups to which this user belongs (`mygroup1` and `mygroup2`):

   ```
   openssl req -new -key myuser.key -out myuser.csr -subj
   "/CN=myuser/O=mygroup1/O=mygroup2"
   ```

3. Create a manifest for the CertificateSigningRequest object and save it to a file (`csr.yaml` in this example):

   > **i** In the `request` field, replace `<CSR_BASE64>` with the contents of `myuser.csr` encoded in Base64 without line breaks, for example, the output of `base64 < myuser.csr | tr -d '\n'` .

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: demo-client-cert
spec:
  request: <CSR_BASE64>
  signerName: "kubernetes.io/kube-apiserver-client"
  expirationSeconds: 7200
  usages:
  - "digital signature"
  - "client auth"
```

4. Apply the manifest to create a certificate signing request:

   ```
   d8 k apply -f csr.yaml
   ```

5. Check that the certificate has been approved and issued:

   ```
   d8 k get csr demo-client-cert
   ```

   If the certificate is issued, it will have the value `Approved,Issued` in the `CONDITION` column. Example output:

   ```
    NAME               AGE     SIGNERNAME                                REQUESTOR
   REQUESTEDDURATION   CONDITION
    demo-client-cert   8m24s   kubernetes.io/kube-apiserver-client   kubernetes-
   admin   120m                Approved,Issued
   ```

   If the certificate is not automatically verified, verify it:

   ```
   d8 k certificate approve demo-client-cert
   ```

   Then, confirm that the certificate has been successfully approved.

6. Extract the encoded certificate from the CSR named `demo-client-cert`, decode it from Base64 and save it to a file (`myuser.crt` in this example):

   ```
   d8 k get csr demo-client-cert -ojsonpath="{.status.certificate}" | base64 -d >
   myuser.crt
   ```

Now the certificate can be specified in the config file:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
    server: https://<cluster_host>:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: myuser
  name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Granting access to the created user

> **i** The example uses the basic role-based model.

To grant access to the created user, create a ClusterAuthorizationRule.

Example of a `ClusterAuthorizationRule`:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: myuser
spec:
  subjects:
  - kind: User
    name: myuser
  accessLevel: PrivilegedUser
  portForwarding: true
```

Configuring kube-apiserver for multi-tenancy mode

The multi-tenancy mode, which allows you to restrict access to namespaces, is enabled by the enableMultiTenancy module's parameter.

Working in multi-tenancy mode requires enabling the Webhook authorization plugin and configuring a `kube-apiserver.` All actions necessary for the multi-tenancy mode are performed **automatically** by the control-plane-manager module; no additional steps are required.

Multi-tenancy mode is available in the editions listed in the parameter description. The API server is configured automatically while the `controlPlaneConfigurator.enabled` parameter is `true`.

Changes to the `kube-apiserver` manifest that will occur after enabling multi-tenancy mode:

- The `--authorization-mode=Node,RBAC` flag is replaced with the structured authorization configuration (`--authorization-config=/etc/kubernetes/deckhouse/extra-files/authorization-config.yaml`) with the `Node`, `Webhook`, and `RBAC` authorizers;
- The `kubeconfig` for the authorization webhook will be created at `/etc/kubernetes/deckhouse/extra-files/webhook-config.yaml`. Both files are in the `/etc/kubernetes/deckhouse/extra-files` directory, which is already mounted into `kube-`

`apiserver`.

How do I check that a user has access?

Execute the command below with the following parameters:

- `resourceAttributes` (the same as in RBAC) - target resources;
- `user` - the name of the user;
- `groups` - user groups;

> **i** If this module is used together with the `user-authn` module, you can find the groups and the username of a user in the Dex logs (`d8 k -n d8-user-authn logs -l app=dex`). The logs have them only after the user has signed in.

```
cat  <<EOF | 2>&1 d8 k  create --raw
/apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "dev",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "user@example.com",
    "groups": [
      "developers"
    ]
  }
}
EOF
```

You will see if access is allowed and what role is used:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by RoleBinding \"<BINDING_NAME>/dev\" of ClusterRole
\"<ROLE_NAME>\" to User \"user@example.com\""
}
```

The answer of the API server already includes the decision of the authorization webhook. To see the answer of the webhook alone, run the following command as root on a master node:

```
cat <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-
config.yaml create --raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "dev",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "user@example.com",
    "groups": [
      "developers"
    ]
  }
}
EOF
```

```
{
  "allowed": false
}
```

The `allowed: false` message means that the webhook doesn’t block access. In case of webhook denying the request, you will see, e.g., the following message:

```
{
  "allowed": false,
  "denied": true,
  "reason": "either you have no access to the namespace or the namespace does not
exist"
}
```

Customizing rights of high-level roles

If you want to grant more privileges to a specific high-level role, you only need to create a ClusterRole with the `user-authz.deckhouse.io/access-level: <AccessLevel>` annotation.

The module automatically sets the `user-authz.deckhouse.io/access-level` label on such ClusterRoles and includes their rules in the aggregated ClusterRole `user-authz:<access-level>:custom`. The label is internal: the module keeps it equal to the annotation and removes it when the annotation is absent, so there is no need to set it by hand. In audit logs, access granted through an annotated ClusterRole is attributed to that aggregated role.

The annotation accepts `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, and `ClusterAdmin`. Other values, including `SuperAdmin`, are ignored. In the name of the aggregated role, the level is written in kebab case, for example, `user-authz:cluster-editor:custom`. The rules of a level are also included in the aggregated roles of every higher level.

An example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

The module lifecycle stage: General Availability

How do I create a user?

Creating a user.

How do I limit user rights to specific namespaces?

To limit a user's rights to specific namespaces in the granular role-based model, use RoleBinding with the namespace role that has the appropriate level of access. See the example of assigning administrative rights to a user within a namespace.

In the basic role-based model, use the `namespaceSelector` or `limitNamespaces` (deprecated)

parameters in the ClusterAuthorizationRule CR. These parameters require the `enableMultiTenancy` parameter: while it is disabled, a ClusterAuthorizationRule with them is refused. A rule that already has them is applied without the limits, see Why is a rule reported as needing multi-tenancy?

What if there are two ClusterAuthorizationRules matching a single user?

In the example, the user `jane.doe@example.com` is in the `administrators` group. There are two cluster authorization rules:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: jane
spec:
  subjects:
    - kind: User
      name: jane.doe@example.com
  accessLevel: User
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
  - kind: Group
    name: administrators
  accessLevel: ClusterAdmin
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: env
        operator: In
        values:
        - prod
        - stage
```

1. `jane.doe@example.com` has the right to get and list any objects in the namespaces labeled `env=review`
2. `Administrators` can get, edit, list, and delete objects at the cluster level and in the namespaces labeled `env=prod` and `env=stage`.

Because `Jane Doe` matches two rules, the resulting permissions are calculated as follows:

- `Jane Doe` will have the highest accessLevel across all matching rules — `ClusterAdmin`.
- The `namespaceSelector` options will be combined, so that Jane will have access to all the namespaces with the `env` label set to one of the following values: `review`, `stage`, or `prod`.

> ⚠ If there is a rule without the `namespaceSelector` option and `limitNamespaces` deprecated option, it means that all namespaces are allowed excluding system namespaces, which will affect the resulting limit namespaces calculation.

Can the basic and the granular role-based models be used at the same time?

Yes. Both models ultimately boil down to the standard Kubernetes RBAC mechanism, and RBAC is a permissive model: permissions from all sources are **summed up**. If an action is allowed by at least one source — a ClusterAuthorizationRule, an AuthorizationRule, a RoleBinding to a granular-model role, or a ProjectRoleBinding — it will be allowed. Nothing needs to be “switched over”: you can keep the existing ClusterAuthorizationRule objects and gradually add granular-model role bindings.

This holds in the multitenancy mode (`enableMultiTenancy`) as well, with one rule about levels. The namespace restriction of a ClusterAuthorizationRule (`limitNamespaces` or `namespaceSelector`) limits the permissions of that rule only: in a namespace outside the list the user keeps whatever a RoleBinding, an AuthorizationRule or a ProjectRoleBinding grants there, at the level of that binding — the `accessLevel` of the ClusterAuthorizationRule does not leak into it. A namespace where the user has no source at all stays denied. See the module description for the full set of sources and an example.

How do I get an equivalent of the ClusterAdmin and SuperAdmin roles in the granular model?

There is no single-object counterpart of the basic model's `ClusterAdmin` and `SuperAdmin` roles in the granular model — it deliberately separates platform administration (system roles) from application access (namespace and project roles). The equivalent is assembled from **two bindings**: a ClusterRoleBinding to a system role and a ClusterProjectRoleBinding to a project role (the latter applies in all projects, including those created later).

Approximate level mapping:

Basic model role	Granular model equivalent
`User`	`d8:namespace:viewer` (via RoleBinding or ProjectRoleBinding).
`PrivilegedUser`	`d8:namespace:user`.
`Editor`	`d8:namespace:manager` — the same level; the scope differs: the granular role is bound per namespace (RoleBinding) or per project (ProjectRoleBinding), not cluster-wide with a namespace filter.
`Admin`	`d8:namespace:admin`.
`ClusterEditor`	No direct counterpart. `ClusterEditor` is the `Editor` level in every namespace, system ones included, plus cluster-scoped objects. Assemble it from a ClusterProjectRoleBinding to `d8:project:manager` (all projects) and a system role for the platform part; `d8:system:manager` alone is not an equivalent: it gives no access to the user namespaces.
`ClusterAdmin`	`d8:system:manager` + a ClusterProjectRoleBinding to `d8:project:admin`.
`SuperAdmin`	`d8:system:superadmin` + a ClusterProjectRoleBinding to `d8:project:superadmin`.

An example for `ClusterAdmin` (the `k8s-admins` group):

```
# Platform: Deckhouse Platform module configuration, cluster-wide resources, system
namespaces.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: k8s-admins-platform
subjects:
  - kind: Group
    name: k8s-admins
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:system:manager
  apiGroup: rbac.authorization.k8s.io
---
# Applications: administrator in all namespaces of all projects (including future
ones).
apiVersion: deckhouse.io/v1alpha3
kind: ClusterProjectRoleBinding
metadata:
  name: k8s-admins-projects
spec:
  subjects:
    - kind: Group
      name: k8s-admins
  roleRef:
    kind: ClusterRole
    name: d8:project:admin
```

For `SuperAdmin`, replace the roles with `d8:system:superadmin` and `d8:project:superadmin`.

Specifics:

- With the `multitenancy-manager` module enabled, every user namespace is a project, so the “system role + ClusterProjectRoleBinding” pair covers both the platform and all user namespaces. Only the `default` namespace is not covered — it is neither a project nor a system namespace.
- You cannot create a custom “all permissions” role (`apiGroups: ["*"], resources: ["*"], verbs: ["*"]`): such a role gives, among other things, project management permissions and is rejected by the built-in protections. If you need truly unrestricted access to the whole API (outside the platform role model), use a ClusterRoleBinding to the built-in Kubernetes `cluster-admin` role — only someone who already has such permissions can assign it.

How do I grant a user access to the resources of one module only?

A typical request: a user in a namespace should only work with the resources of one module (for example, only with virtual machines) without seeing the other resources (Pod, Deployment, etc.).

Every Deckhouse Platform module ships separate capabilities for its resources, so such access is granted without writing RBAC rules. Assemble a custom role that aggregates only the capabilities of the desired module. Select each capability by its `rbac.deckhouse.io/capability` label: a role with the `rbac.deckhouse.io/delegatable: "true"` label may select capabilities only by this label or by the `rbac.deckhouse.io/aggregate-to-namespace-as` and `rbac.deckhouse.io/aggregate-to-project-as` labels, and a selector by `module` or by any other label is refused. To list the capabilities of the module, run:

```
d8 k get clusterroles -l
rbac.deckhouse.io/kind=capability,module=virtualization,rbac.deckhouse.io/scope=namespace -o custom-
columns='NAME:.metadata.name,CAPABILITY:.metadata.labels.rbac\.deckhouse\.io/capabili
ty'
```

An example of the role, where `<ACTION>` is to be replaced with the values from the list:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace:virtualization-only
  labels:
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/delegatable: "true"   # Allows using the role in a RoleBinding
inside projects.
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/capability: namespace-capability.virtualization.<ACTION>
    - matchLabels:
        rbac.deckhouse.io/capability: namespace-capability.virtualization.<ACTION>
rules: []
```

Values without the `custom.` prefix are reserved for the capabilities Deckhouse ships, and a ClusterRole a user writes with such a value is refused, so these selectors pick only the capabilities of the module.

Grant the role via a RoleBinding in the desired namespace or via a ProjectRoleBinding across the whole project. The user will get access only to the module's resources — the standard Kubernetes resources will not be visible to them.

> **i** Do not bind the capability of the module directly. Inside **project** namespaces a plain RoleBinding may only reference roles available to the project, and capabilities are not among them by default. The custom role above has the `rbac.deckhouse.io/delegatable: "true"` label, which makes it available there, so the same role works in every namespace.

What happens to the bindings when the cluster is updated to the release with the controller?

Nothing is recreated and access is not interrupted. In Deckhouse Platform 1.78, the `user-authz-controller` component adopts the bindings that the Helm chart used to render: the same objects with the same names, with an `ownerReference` to the rule added.

Plan for the release that performs the migration: it processes every existing binding once, so it takes longer than usual. Subsequent releases are not affected.

Bindings in the release	What to expect
Up to 5000	The release runs longer than usual and completes on its own
More than 5000	The release can exceed the 20-minute module timeout and fail

Count the bindings before you update. To get the number of cluster role bindings, use the following command:

```
d8 k get clusterrolebindings -l heritage=deckhouse,module=user-authz --no-headers | wc -l
```

To get the number of namespaced bindings, use the following command:

```
d8 k get rolebindings -A -l heritage=deckhouse,module=user-authz --no-headers | wc -l
```

If there are more than 5000, raise the release timeout or switch to the previous release engine for the duration of the update, and revert the setting afterwards:

To increase the release timeout, use the following command:

```
d8 k -n d8-system set env deploy/deckhouse HELM_TIMEOUT=60m
```

To switch back to the previous engine, use the following command:

```
d8 k -n d8-system set env deploy/deckhouse USE_NELM=false
```

If the migration cannot prepare every binding, the release does not start and the bindings stay as they were.

After the migration, the release no longer grows with the number of rules. The `D8UserAuthzReleaseSlow` alert fires when applying the module release repeatedly takes more than 5 minutes, and `D8UserAuthzRenderSlow` fires when rendering its templates repeatedly takes more than 30 seconds. Both point at the API server or at a Deckhouse Pod short of CPU rather than at the rules.

How do I check that the controller keeps the bindings in sync?

The `user-authz-controller` component reconciles the ClusterRoleBinding and RoleBinding objects of every ClusterAuthorizationRule and AuthorizationRule, the `d8:dict` ClusterRoleBindings of the granular model, and the RoleBinding objects that give the subjects of system and subsystem roles the namespace role named by `rbac.deckhouse.io/use-role` in module namespaces. Its state is shown by the object status, the metrics, and the alerts.

Object status. Every rule carries a `Ready` condition and the number of its bindings:

To list all ClusterAuthorizationRules in the cluster, run:

```
d8 k get clusterauthorizationrules
```

Example output:

```
NAME        ACCESS LEVEL   READY   BINDINGS   AGE
my-rule     Admin          True    3          5d
```

To list all AuthorizationRules in all namespaces of the cluster, run:

```
d8 k get authorizationrules -A
```

Example output:

```
NAMESPACE   NAME       ACCESS LEVEL   READY   BINDINGS   AGE
default     app-rule   Editor         True    2          3d
team-a      dev-rule   User           False   0          1h
```

To get only the conditions of a specific ClusterAuthorizationRule, run:

```
d8 k get clusterauthorizationrule <name> -o jsonpath='{.status.conditions}'
```

Example output:

```
[
  {
    "lastTransitionTime": "2026-09-11T10:00:00Z",
    "message": "3 bindings applied",
    "observedGeneration": 1,
    "reason": "BindingsApplied",
    "status": "True",
    "type": "Ready"
  }
]
```

`Ready=True` with the reason `BindingsApplied` means the bindings match the rule. `Ready=False` indicates the problem: `InvalidSpec` (the rule cannot be rendered into bindings; the message says why) or `ApplyError` (the API server rejected a write; the message names the binding and the error). Events with the same reasons are recorded on the rule.

Metrics. The controller exposes them on the `metrics` port of its Pod and the `PodMonitor` `user-authz-controller` collects them (the `operator-prometheus` module must be enabled). The `kind` label is `ClusterAuthorizationRule`, `AuthorizationRule`, `dict` or `manage`.

Metric	Description
`d8_user_authz_authorization_rules{kind}`	Objects of the kind known to the controller
`d8_user_authz_bindings_desired{kind}`	Bindings the objects of the kind must have
`d8_user_authz_bindings_actual{kind}`	Bindings of the kind that exist
`d8_user_authz_bindings_drift{kind,reason}`	Bindings not in the desired state at the last reconcile; `reason` is `missing`, `extra` or `changed`. Stays above zero only while the controller fails to converge
`d8 user authz_bindings_apply_total{kind,op,result}`	Write operations issued by the controller (`op`: `create`, `update`, `delete`; `result`: `success`, `error`)
`d8 user authz_authorization_rules_invalid{kind,reason}`	Rules whose `Ready` condition is `False`, by kind and reason
`d8 user authz authorization rule_invalid{kind,name,rule_namespace,reason}`	`1` for every such rule (at most 50 rules are named; the aggregate above is always complete)
`controller runtime reconcile total`, `controller_runtime_reconcile_errors_total`, `controller_runtime_reconcile_time_seconds`	Standard controller-runtime metrics per reconciler (`controller` label: `clusterauthorizationrule-bindings`, `authorizationrule-bindings`, `dict-bindings`, `manage-bindings`)

Deckhouse exports the following metrics of the module hooks:

Metric	Description
`d8_user_authz_custom_cluster_roles{level}`	Custom `ClusterRole` objects (annotated with `user-authz.deckhouse.io/access-level`) per access level
`d8 user_authz_custom_aggregation_missing{level}`	`1` when the aggregated ClusterRole `user-authz:<level>:custom` has no rules although custom roles it must aggregate exist
`d8_user_authz_bindings_keep_stamped_total`, `d8_user_authz_keep_stamp_duration_seconds`	Work of the migration hook that protects chart-rendered bindings from being pruned by the release: objects stamped and the duration of its last run

Alerts. The module ships the following alerts for the controller:

- `D8UserAuthzControllerUnavailable` and `D8UserAuthzControllerTargetDown` when the controller has unavailable replicas or is not scraped for 5 minutes.
- `D8UserAuthzControllerReconcileErrorsHigh` on a sustained reconcile error rate.
- `D8UserAuthzBindingsDrift` when bindings of some kind stay out of the desired state for 15 minutes.
- `D8UserAuthzAuthorizationRulesInvalid` and `D8UserAuthzAuthorizationRuleInvalid` counting and naming the rules whose bindings cannot be applied for 10 minutes.
- `D8UserAuthzCustomRolesNotAggregated` when the aggregated ClusterRole of an access level stays empty although custom roles for it exist.

When a rule stays `Ready=False` with `ApplyError` because a binding's `roleRef` was changed by hand, delete that binding: `roleRef` is immutable and the controller recreates the binding with the right one.

How do I check that the authorization webhook has the current rules?

The authorization webhook and Permission Browser read the multi-tenancy options of ClusterAuthorizationRules (`limitNamespaces`, `namespaceSelector`, `allowAccessToSystemNamespaces`) from the API, so a change reaches them within a few seconds.

To check that the webhook Pods are running, run:

```
d8 k -n d8-user-authz get pods -l app=user-authz-webhook -o wide
```

Example output (one Pod per master, on the host network; both containers must be ready):

```
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
user-authz-webhook-qrm2n      2/2     Running   0          2d    10.0.0.11    master-0
<none>           <none>
user-authz-webhook-h6jbh      2/2     Running   0          2d    10.0.0.12    master-1
<none>           <none>
user-authz-webhook-97fvs      2/2     Running   0          2d    10.0.0.13    master-2
<none>           <none>
```

To see the last 100 log lines of the webhook container of every such Pod, run:

```
d8 k -n d8-user-authz logs -l app=user-authz-webhook -c webhook --tail=100
```

Example output:

```
2026/09/11 10:00:00 server is starting to listen on  127.0.0.1:40443 ...
2026/09/11 10:00:01 rules source: directory rebuilt from 12 rules (18 subjects, 0
quarantined) in 4.2ms
```

The `rules source: directory rebuilt from N rules` line shows the informer has listed the rules and how many it holds; `quarantined` counts the rules the instance could not fully use: a `limitNamespaces` pattern or a `namespaceSelector` that does not compile, or a rule that cannot be read.

An instance that has not read the rules yet is not ready and answers authorization requests with an error instead of a denial. In the seconds between a rule being created and the webhook observing it, the subjects of that rule that no observed rule covers yet are denied access to all namespaces. A subject that another rule already covers keeps the scope of that rule.

The webhook exposes metrics through a `kube-rbac-proxy` sidecar; the PodMonitor `user-authz-webhook` collects them, and the `operator-prometheus` module must be enabled. There is one series per master node.

Metric	Description
`user_authz_webhook_rules_informer_synced`	`1` once the instance has read the ClusterAuthorizationRules at least once
`user_authz_webhook_rules_observed`	Rules the instance currently uses
`user_authz_webhook_rules_subjects`	Distinct subjects in those rules
`user_authz_webhook_rules_max_resource_version`	Highest `resourceVersion` among the observed rules. Compare it with the cluster to measure the lag
`user_authz_webhook_rules_quarantined`	Rules the instance could not fully use: a `limitNamespaces` pattern or `namespaceSelector` that does not compile, or a rule that cannot be read at all
`user_authz_webhook_rules_directory_updated_timestamp_seconds`	Time of the last rules update
`user_authz_webhook_rules_directory_rebuilds_total`, `user_authz_webhook_rules_directory_rebuild_duration_seconds`	Number of rules updates and their duration
`user_authz_webhook_rules_watch_errors_total`	List and watch errors of the rules informer

Permission Browser exposes the same metrics with the `user_authz_permission_browser` prefix; the PodMonitor `permission-browser-apiserver` collects them.

The module ships the following alerts for both components:

Alert	Fires when

Alert	Fires when
`D8UserAuthzWebhookTargetDown`	Prometheus has not scraped at least one webhook instance for 5 minutes
`D8UserAuthzWebhookRulesQuarantined`	A rule has not compiled for 10 minutes
`D8UserAuthzWebhookRulesWatchErrors`	The rules informer has been failing to watch the rules for 10 minutes
`D8UserAuthzWebhookDirectoryDiverged`	Instances have been using different sets of rules for 10 minutes, so the same request is answered differently depending on the master node that receives it
`D8UserAuthzRulePropagationLag`	One instance has not updated its rules for an hour while another one has
`D8UserAuthzPermissionBrowserUnavailable`	Permission Browser has unavailable replicas

Why does a change to a ClusterAuthorizationRule take up to 30 seconds to take effect?

The API server caches the answers of the authorization webhook. In the `AuthorizationConfiguration` that `control-plane-manager` renders, the webhook has `authorizedTTL: 5m`, `unauthorizedTTL: 30s` and `timeout: 3s`. The cache key is the whole SubjectAccessReview, so an identical repeated request is answered from the cache.

The webhook never allows a request: it either denies it or gives no opinion, and both answers are cached for `unauthorizedTTL`, which is 30 seconds. Creating a rule or narrowing an existing one therefore takes effect for a particular request within 30 seconds of the last identical request:

```
d8 k auth can-i --as=user@example.com get pods -n other-namespace
sleep 30
d8 k auth can-i --as=user@example.com get pods -n other-namespace
```

The cache cannot be flushed without restarting `kube-apiserver`.

For the resource of a CRD installed a moment ago, the delay reaches 40 seconds. The webhook learns whether a resource is namespaced from discovery, which it queries no more often than once every 10 seconds, and the API server then caches the answer for another 30 seconds.

During those seconds the namespace limits do not apply to that resource: the webhook does not know it exists, gives no opinion, and RBAC alone answers — so a subject whose rule limits it to some namespaces can list the new resource across the whole cluster. Installing a CRD requires far more privilege than that yields, which is why discovery is queried at a bounded rate rather than on every request. Permission Browser has a window of the same order for its own report, up to 30 seconds.

Why is a rule reported as needing multi-tenancy?

The `limitNamespaces`, `namespaceSelector` and `allowAccessToSystemNamespaces` options are enforced by the authorization webhook, which is deployed only when `enableMultiTenancy` is enabled. A rule that uses these options while multi-tenancy is disabled is applied without them: the subjects named in the rule hold its access level in **every** namespace of the cluster, including the system ones.

While multi-tenancy is disabled, the `d8-user-authz-car-multitenancy-related-options.deckhouse.io` admission webhook refuses a ClusterAuthorizationRule with any of these options, even `allowAccessToSystemNamespaces: false`. It also refuses disabling `enableMultiTenancy` while a rule uses them. So a rule reaches this state only if it was written before these checks applied, for example before an update. Any change to such a rule must also remove the options.

Only options that ask for a limit are counted. A rule with `allowAccessToSystemNamespaces: false`, or with an empty `limitNamespaces` list, is not reported.

Two metrics report the situation:

Metric	Description
`d8_user_authz_rule_needs_multitenancy{name,options}`	One series per affected rule, up to 50, with the options that will not take effect

Metric	Description
`d8_user_authz_rules_needing_multitenancy`	Total number of affected rules, including those past the first 50

An individual rule is listed in the `D8UserAuthzRuleNeedsMultiTenancy` alert, and `D8UserAuthzRulesNeedMultiTenancy` fires when there are more affected rules than the first alert lists. Both belong to the `D8UserAuthzMisconfigured` group.

To resolve the situation, either enable multi-tenancy, or remove the options from the rules.

To check whether multi-tenancy is enabled, use the following command:

```
d8 k get moduleconfig user-authz -o jsonpath='{.spec.settings.enableMultiTenancy}'
```

To retrieve the names of ClusterAuthorizationRules that restrict access by namespace or allow access to system namespaces, use the following command:

```
d8 k get clusterauthorizationrule -o json | jq -r '.items[] | select((.spec.limitNamespaces
// [] | length > 0) or (.spec.namespaceSelector != null) or
(.spec.allowAccessToSystemNamespaces == true)) | .metadata.name'
```

How do I extend a role or create a new one?

The granular role model is based on the aggregation principle; it compiles smaller roles into larger ones, thus providing easy ways to enhance the model with custom roles.

Creating a new role subsystem

Suppose that the current subsystems do not fit the role distribution in the company. You need to create a new subsystem that includes roles from the `cluster` subsystem, the `network` subsystem and the user-authn module.

To meet this need, create the following role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: viewer
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-cluster-as: manager
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-network-as: manager
    - matchLabels:
        rbac.deckhouse.io/scope: system
        module: user-authn
rules: []
```

The labels for the new role listed at the top suggest that:

- The namespace role `user-authz-controller` gives the subjects of the role in the module namespaces; a custom role is projected as `viewer` at most:

  ```
  rbac.deckhouse.io/use-role: viewer
  ```

- The role is a custom role (custom roles never define their own rules, they only aggregate capabilities):

```
rbac.deckhouse.io/kind: custom-role
```

> **i** This label is mandatory.

- The role is a subsystem one, and it shall be handled accordingly:

```
rbac.deckhouse.io/scope: subsystem
```

- There is a subsystem for which the role is responsible:

```
rbac.deckhouse.io/subsystem: mycustom
```

Then there are selectors that implement aggregation:

- This one aggregates the manager role from the `cluster` subsystem:

```
rbac.deckhouse.io/aggregate-to-cluster-as: manager
```

- This one aggregates all the system-scope capabilities defined for the `user-authn` module:

```
rbac.deckhouse.io/scope: system
module: user-authn
```

This way, your role will combine the permissions of the `cluster` subsystem, `network` subsystem, and the `user-authn` module.

Notes:

- Custom roles and capabilities must be named with the `d8:custom:` prefix (the rest of the `d8:` prefix space is reserved for Deckhouse Platform built-in objects). The name must agree with the declared scope: a subsystem role is `d8:custom:<subsystem>:<name>` (the segment is the subsystem itself, as in the example above), a namespace or project role is `d8:custom:namespace:<name>` or `d8:custom:project:<name>`, and a capability is `d8:custom:<scope>-capability:<name>`. A name that disagrees with the `rbac.deckhouse.io/scope` label is rejected.
- RoleBindings with a namespace role (`d8:namespace:<level>`) will be created in the namespaces of the aggregated subsystems' modules, the level is specified by the `rbac.deckhouse.io/use-role` label, `viewer` at most for a custom role. Without the label no such RoleBinding is created.
- A custom role cannot be aggregated into `d8:system:<level>`, because only a custom capability may carry the `rbac.deckhouse.io/aggregate-to-system-as` label. To give the role to the holders of `d8:system:manager`, bind it to them with a ClusterRoleBinding.

Extending the custom role

Suppose a new cluster CRD object, MySuperResource, has been created in the cluster (a manage role example), and you need to extend the custom role from the example above to include the permissions to interact with this resource.

First, you have to add a new selector to the role:

```
rbac.deckhouse.io/aggregate-to-mycustom-as: manager
```

This selector would enable capabilities to be aggregated to a new subsystem by specifying this label. After adding the new selector, the role will look as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: viewer
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-cluster-as: manager
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-network-as: manager
    - matchLabels:
        rbac.deckhouse.io/scope: system
        module: user-authn
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-mycustom-as: manager
rules: []
```

Next, you need to create a new capability and define permissions for the new resource, e. g., the read-only permission:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-mycustom-as: manager
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
    rbac.deckhouse.io/capability: "custom.subsystem-
capability.mycustom.superresource_view"
  name: d8:custom:subsystem-capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

The capability will add its permissions to the subsystem role. so that holders of the role will be able to view the new object.

Notes:

- Name a custom capability `d8:custom:<SCOPE>-capability:<NAME>`, where `<SCOPE>` is the value of its `rbac.deckhouse.io/scope` label. A name that disagrees with the label is rejected.

Extending the existing subsystem roles

To extend an existing role, follow the procedure outlined in the section above. Be sure to change the labels and the role name!

For example, here’s how you can extend the manager role from the `cluster` (`d8:subsystem:cluster:manager`) subsystem:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-cluster-as: manager
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: cluster
    rbac.deckhouse.io/capability: "custom.subsystem-
capability.cluster.superresource_view"
  name: d8:custom:subsystem-capability:cluster:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This way, the new capability will extend the `d8:subsystem:cluster:manager` role.

Extending subsystem roles and adding a new namespace

The namespaces where the holders of a system or subsystem role get a RoleBinding with the namespace role are the namespaces of its modules: `user-authz-controller` takes them from the `rbac.deckhouse.io/namespace` label of the system capabilities Deckhouse ships, which have a `d8:` name and the `heritage: deckhouse` label. On a custom capability the label is ignored: the capability adds its rules to the roles that aggregate it, but no namespace.

To give the subjects of such a role access to another namespace, bind them to a namespace role there with a RoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cluster-managers-viewer
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: d8:namespace:viewer
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: cluster-managers
```

Extending the existing namespace roles

If the resource belongs to a namespace, you need to extend the namespace role instead of the system/subsystem role. The only difference is the labels and the name:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-namespace-as: user
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/capability: "custom.namespace-
capability.mycustom.superresource_view"
  name: d8:custom:namespace-capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This capability will be added to the `d8:namespace:user` role.

Creating a custom namespace or project role

Sometimes the built-in hierarchy of levels does not fit: for example, you need a “developer” role — full view of the namespace plus terminal access to pods (`exec`, `attach`), but without the right to change quotas or RBAC. Such a role is assembled from ready-made capabilities, without writing RBAC rules by hand.

The rules for custom roles:

- the name must start with `d8:custom:` (for example, `d8:custom:namespace:developer`);
- the role must carry the `rbac.deckhouse.io/kind: custom-role` label;
- a namespace or project role that will be used in a RoleBinding must also carry `rbac.deckhouse.io/delegatable: "true"`. Every user namespace is a project, and a RoleBinding there is accepted only for roles with this label. Do not put it on system or subsystem roles — the admission webhook rejects that;
- the role **cannot contain its own rules** (`rules`) — it may only aggregate capabilities via `aggregationRule`. Permissions are described in separate capabilities, so the contents of the role stay transparent;
- a single role cannot aggregate capabilities of the user-facing scopes (`namespace`, `project`) together with the administrative ones (`system`, subsystems) — such a role is rejected.

An example: a role that includes everything allowed by `d8:namespace:viewer`, plus one specific capability (connecting to pods), selected precisely by its unique `rbac.deckhouse.io/capability` label:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace:developer
  labels:
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/delegatable: "true"   # Required for a RoleBinding inside a
project namespace.
  annotations:
    custom.meta.deckhouse.io/title: "Developer"
    custom.meta.deckhouse.io/description: "View resources and connect to pods,
without managing quotas and RBAC"
aggregationRule:
  clusterRoleSelectors:
    # Everything included in the viewer level of the namespace lineage.
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-namespace-as: viewer
    # Plus one specific capability selected by its unique name.
    - matchLabels:
        rbac.deckhouse.io/capability: "namespace-
capability.kubernetes.access_terminal"
rules: []
```

If no existing capability grants the permissions you need, create your own (a `custom-capability` may contain rules) and add a selector by its `rbac.deckhouse.io/capability` label to the role's `aggregationRule` (in the example below — `matchLabels: {rbac.deckhouse.io/capability: "custom.namespace-capability.port-forward"}`). The label value of a custom capability is `custom.<SCOPE>-capability.<NAME>`, and a role bindable in a project may select only the `namespace-capability` and `project-capability` scopes:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace-capability:port-forward
  labels:
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/capability: "custom.namespace-capability.port-forward"
rules:
  - apiGroups: [""]
    resources: ["pods/portforward"]
    verbs: ["create"]
```

To get a list of all available capabilities and their unique names, use the command:

```
d8 k get clusterroles -l rbac.deckhouse.io/kind=capability -o custom-
columns='NAME:.metadata.name,CAPABILITY:.metadata.labels.rbac\.deckhouse\.io/capabili
ty'
```

The created role is assigned exactly like a built-in one: via a RoleBinding in a namespace or via a ProjectRoleBinding across a whole project (for project roles, use `rbac.deckhouse.io/scope: project` and aggregate `aggregate-to-project-as`). A RoleBinding in a project namespace requires the `delegatable` label on the role; a ProjectRoleBinding does not (it has its own prefix allow-list). The role cannot be assigned via a ClusterRoleBinding — just like the built-in roles of these scopes.

> **i** You can also assemble such a role without YAML — with the access grant wizard in the Deckhouse web UI: it shows the available capabilities, builds a role out of them, and immediately creates the required binding.

How do I migrate custom roles to the new scheme?

Along with the role renaming, the label scheme that drives aggregation has changed.

> ⚠ The built-in capabilities no longer carry the `rbac.deckhouse.io/kind: manage` and `rbac.deckhouse.io/kind: use` labels. A custom role of the old scheme that selects capabilities by these labels has lost their permissions, and so have its subjects. Custom roles get no compatibility aliases, so repair such roles as described below.

An aggregation selector of a custom role that requires one of these labels (for example, `rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-<SUBSYSTEM>-as`) matches none of the built-in capabilities. From such a selector, the Kubernetes aggregation controller takes only the rules of the custom capabilities that still carry the old labels. Deckhouse Platform creates the RoleBinding objects for the subjects of a role with the `rbac.deckhouse.io/use-role` label in the namespaces of the modules only when the role has the `rbac.deckhouse.io/scope: system` or `subsystem` label. A custom role of the old scheme with `use-role` and without that scope has lost these objects, even if its selectors are already rewritten. A selector that requires the aggregation label of a lineage that no built-in role collects matches none of the built-in capabilities either, because the modules label their capabilities only with the lineages that the built-in roles collect. Such a label is, for example, `rbac.deckhouse.io/aggregate-to-all-as` of the old scheme, the label of a subsystem the role model no longer has, or the label of a subsystem a module has left. A lineage that a custom capability carries is an exception, because the selector still gets the rules of that capability. A custom capability here is a ClusterRole without the `heritage: deckhouse` label and without an `aggregationRule` field. So is the lineage of the role's own subsystem, from its `rbac.deckhouse.io/subsystem` label, because the custom capabilities of that subsystem carry it (for details, see Extending the custom role). Neither exception applies to the `all`, `deckhouse`, `infrastructure`, `kubernetes` and `networking` lineages, which do not count as custom ones. A compatibility alias of an old role, a ClusterRole with the `rbac.deckhouse.io/deprecated: "true"` label, does not count as a built-in role that collects a lineage, because it collects the lineage of an old subsystem only for the bindings to the old name. A custom capability of the old scheme may also have moved to other roles (for details, see Replacing a capability).

Such roles are listed in the `D8UserAuthzLegacyRBACv2CustomRoleFound` alert. It fires for every ClusterRole without the `heritage: deckhouse` label and with an `aggregationRule` field that has a selector requiring `rbac.deckhouse.io/kind: manage` or `use`, has a selector requiring such an aggregation label, or carries `rbac.deckhouse.io/use-role` without `rbac.deckhouse.io/scope: system` or `subsystem`, whatever the name of the role. A role without `aggregationRule`, such as a role frozen earlier, is not listed. To list such roles, use the command:

```
d8 k get clusterroles -o json | jq -r '
  def lineages: [((.matchLabels // {}) | keys[]), (.matchExpressions[]? |
select(.operator == "In" or .operator == "Exists") | .key)]
    | map(capture("^rbac\\.deckhouse\\.io/aggregate-to-(?<lineage>.+)-
as$").lineage);
  ([.items[] | select(.metadata.labels.heritage == "deckhouse" and
.metadata.labels["rbac.deckhouse.io/deprecated"] != "true")
        | .aggregationRule.clusterRoleSelectors[]? | lineages[]] + ["system",
"namespace", "project"]) as $collected
  | ([.items[] | select(.metadata.labels.heritage != "deckhouse" and
.aggregationRule == null) | (.metadata.labels // {}) | keys[]
        | capture("^rbac\\.deckhouse\\.io/aggregate-to-(?<lineage>.+)-
as$").lineage]) as $carried
  | .items[]
  | select(.metadata.labels.heritage != "deckhouse" and .aggregationRule != null)
  | (.metadata.labels["rbac.deckhouse.io/subsystem"] // "") as $own
  | select(
      (
        ((.metadata.labels["rbac.deckhouse.io/use-role"] // "") != "")
        and (((.metadata.labels["rbac.deckhouse.io/scope"] // "") | IN("system",
"subsystem")) | not)
      )
      or (
        [
          .aggregationRule.clusterRoleSelectors[]?
          | (.matchLabels["rbac.deckhouse.io/kind"] // empty),
            (.matchExpressions[]? | select(.key == "rbac.deckhouse.io/kind" and
.operator == "In") | .values[])
        ]
        | any(IN("manage", "use"))
      )
      or (
        [
          .aggregationRule.clusterRoleSelectors[]? | lineages[]
          | select((IN($collected[]) | not)
              and (IN("all", "deckhouse", "infrastructure", "kubernetes",
"networking") or (. != $own and (IN($carried[]) | not))))
        ]
        | length > 0
      )
    )
  | .metadata.name'
```

Mapping between the old and the new scheme:

Before (old scheme)	After (new scheme)
Arbitrary role name (for example, `custom:manage:mycustom:manager`)	Mandatory `d8:custom:` prefix (for example, `d8:custom:mycustom:manager`)
`rbac.deckhouse.io/kind: manage` or `use` on your role	`rbac.deckhouse.io/kind: custom-role`
`rbac.deckhouse.io/kind: manage` or `use` on your capability	`rbac.deckhouse.io/kind: custom-capability`, name prefixed with `d8:custom:`
`rbac.deckhouse.io/level: all | subsystem | module`	`rbac.deckhouse.io/scope: system | subsystem | namespace`
`rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-all-as: <level>` on your capability	`rbac.deckhouse.io/aggregate-to-system-as: <level>`

Before (old scheme)	After (new scheme)
`rbac.deckhouse.io/kind: use` + `rbac.deckhouse.io/aggregate-to-all-as: <level>` on your capability	`rbac.deckhouse.io/kind: custom-capability` + `rbac.deckhouse.io/scope: namespace` + `rbac.deckhouse.io/aggregate-to-namespace-as: <level>`, name `d8:custom:namespace-capability:<name>`
`rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-all-as: <level>` on your role	A ClusterRoleBinding of the role to the subjects of `d8:manage:all:<level>` and `d8:system:<level>`, because a custom role cannot carry `rbac.deckhouse.io/aggregate-to-system-as`
`rbac.deckhouse.io/kind: use` + `rbac.deckhouse.io/aggregate-to-kubernetes-as: <level>` on your capability	`rbac.deckhouse.io/kind: custom-capability` + `rbac.deckhouse.io/scope: namespace` + `rbac.deckhouse.io/aggregate-to-namespace-as: <level>`, name `d8:custom:namespace-capability:<name>`
Aggregation selector: `rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-<subsystem>-as: <level>`	Only `rbac.deckhouse.io/aggregate-to-<subsystem>-as: <level>`
Selector for use permissions: `rbac.deckhouse.io/kind: use` + `rbac.deckhouse.io/aggregate-to-kubernetes-as: <level>`	`rbac.deckhouse.io/aggregate-to-namespace-as: <level>`
Per-module selector: `rbac.deckhouse.io/kind: manage` + `module: <module>`	`rbac.deckhouse.io/scope: system` + `module: <module>`
Labels of the `deckhouse`, `infrastructure` and `kubernetes` subsystems (`rbac.deckhouse.io/aggregate-to-deckhouse-as` and the others)	One label of the `cluster` subsystem, `rbac.deckhouse.io/aggregate-to-cluster-as`
Label of the `networking` subsystem, `rbac.deckhouse.io/aggregate-to-networking-as`	`rbac.deckhouse.io/aggregate-to-network-as`
The `user-authz`, `user-authn` and `multitenancy-manager` modules with the labels of the `security` and `deckhouse` subsystems	The label of the `iam` subsystem, `rbac.deckhouse.io/aggregate-to-iam-as`

The names of the built-in capabilities changed as well (no aliases):

- `d8:manage:permission:module:<module>:view|edit` → `d8:system-capability:<module>:view|edit`
- `d8:use:capability:module:<module>:view|edit` → `d8:namespace-capability:<module>:view|edit`

Aggregation selectors match labels, not names. Bind roles, not capabilities.

Fixing a role

Fix every role that is listed in the `D8UserAuthzLegacyRBACv2CustomRoleFound` alert. Keep the order of the steps, so that the old role and its bindings stay in place until the new ones grant the access:

1. Create a role of the new scheme next to the old one, with the `d8:custom:` name prefix, the `rbac.deckhouse.io/kind: custom-role` label, and the aggregation selectors of the new scheme, without `rbac.deckhouse.io/kind` (for details, see the before and after example). Replace a selector by the aggregation label of a lineage that no built-in role collects as described in Selectors by a lineage no built-in role collects. Choose the scope of the role, and name it accordingly:
    - A role with the `rbac.deckhouse.io/use-role` label, or with the `rbac.deckhouse.io/level: subsystem` or `all` label, gets `rbac.deckhouse.io/scope: subsystem` and `rbac.deckhouse.io/subsystem: <SUBSYSTEM>` with the name `d8:custom:<SUBSYSTEM>:<NAME>`, or `rbac.deckhouse.io/scope: system` with the name `d8:custom:system:<NAME>`. Set the `rbac.deckhouse.io/use-role` label on it to `viewer`. For the subjects of its ClusterRoleBinding objects, Deckhouse then creates RoleBinding objects to `d8:namespace:viewer` in the namespaces of the modules whose system capabilities the role aggregates: a custom role is projected as `viewer` at most, and the

webhook refuses a higher value. Without the label the role is not projected into those namespaces at all. The `rbacv2-cluster-roles.deckhouse.io` webhook refuses a custom role without the `rbac.deckhouse.io/scope` label whose selectors name the aggregation label of a subsystem or of the system scope, such as `rbac.deckhouse.io/aggregate-to-deckhouse-as`.
    - A namespace or project role that you bind with a RoleBinding gets `rbac.deckhouse.io/scope: namespace` and `rbac.deckhouse.io/delegatable: "true"` with the name `d8:custom:namespace:<NAME>` (for details, see Creating a custom namespace or project role).
2. Create capabilities of the new scheme in place of the custom capabilities of the old scheme that the role aggregates, as described in Replacing a capability. Keep the old capabilities for now.
3. For every RoleBinding and ClusterRoleBinding object to the old role, create an object with the same subjects to the new role. The `roleRef` field is immutable, so create a new object instead of editing the old one. If the old role has the `rbac.deckhouse.io/kind: manage` and `rbac.deckhouse.io/aggregate-to-all-as` labels, also bind the new role as described in Roles aggregated into the system roles. With `rbac.deckhouse.io/kind: use`, it extended the namespace roles, see the same section.
4. Check the permissions of the subjects:

   Check whether the specified user can perform an action on a resource across the whole cluster (for cluster-wide resources) or in the current default namespace (for namespaced resources):

   ```
   d8 k auth can-i <VERB> <RESOURCE> --as=<USER_NAME>
   ```

   Check whether the specified user can perform an action on a resource in the specified namespace:

   ```
   d8 k auth can-i <VERB> <RESOURCE> -n <NAMESPACE> --as=<USER_NAME>
   ```

   To check the permissions of a group, add `--as-group=<GROUP_NAME>` to `--as`.
5. Delete the old RoleBinding and ClusterRoleBinding objects, the old role, and the old capabilities.

If the objects are applied from a Git repository (for example, by Argo CD or Flux), make the same changes in the repository. Otherwise, the next apply creates the old objects again.

Roles aggregated into the system roles

In the old scheme, the `rbac.deckhouse.io/aggregate-to-all-as: <LEVEL>` label meant one of two things, depending on the kind label next to it. With `rbac.deckhouse.io/kind: use`, it extended the `d8:use:role:<LEVEL>` roles in every namespace. Their counterpart is `d8:namespace:<LEVEL>`, which selects `rbac.deckhouse.io/aggregate-to-namespace-as: <LEVEL>`, so move such permissions into a namespace capability as described in Replacing a capability. With `rbac.deckhouse.io/kind: manage`, a role with this label extended `d8:manage:all:<LEVEL>`, and the rest of this section is about it. No role of the new model selects this label. Only the deprecated `d8:manage:all:<LEVEL>` and `d8:use:role:<LEVEL>` aliases still select it, depending on the kind label next to it, until they are removed. The `d8:manage:all:<LEVEL>` alias and the `d8:system:<LEVEL>` role select `rbac.deckhouse.io/aggregate-to-system-as: <LEVEL>`, and the `rbacv2-cluster-roles.deckhouse.io` webhook admits this label only on a custom capability. A custom role therefore cannot extend these roles, and the subjects that got the old role through them get the new role only through a ClusterRoleBinding of their own.

The following commands create such a ClusterRoleBinding for each ClusterRoleBinding to `d8:manage:all:<LEVEL>` and `d8:system:<LEVEL>`:

```
NEW_ROLE=<NEW_ROLE_NAME>
LEVEL=<LEVEL>
d8 k get clusterrolebindings -o json > clusterrolebindings.json
jq --arg role "$NEW_ROLE" --arg level "$LEVEL" '
  {apiVersion: "v1", kind: "List", items: [.items[]
    | select(.roleRef.kind == "ClusterRole" and (.roleRef.name | IN("d8:manage:all:"
+ $level, "d8:system:" + $level)))
    | {apiVersion, kind, metadata: {name: ($role + ":" + .metadata.name)},
       roleRef: {apiGroup: "rbac.authorization.k8s.io", kind: "ClusterRole", name:
$role}, subjects}]}' \
  clusterrolebindings.json > new-clusterrolebindings.json
d8 k apply -f new-clusterrolebindings.json
```

Review `new-clusterrolebindings.json` before you create the objects, and remove the subjects that must not get the role. Deckhouse Platform does not keep these objects in sync with the ClusterRoleBinding objects they are copied from. When you revoke the access of a subject, delete both. The name of a copy ends with the name of its source.

Selectors by a lineage no built-in role collects

A selector by the aggregation label of a lineage selected the capabilities that carried this label. For example, `rbac.deckhouse.io/aggregate-to-virtualization-as: <LEVEL>` selected the capabilities of the `virtualization` module. Once the module labels them with the lineage of the subsystem it is in now, a selector by that lineage selects the capabilities of every module of the subsystem, which can be much more than the role gave before. To keep the new role as narrow as the old one, select the capabilities themselves by the `rbac.deckhouse.io/capability` label. Its value does not depend on the subsystem. It is `system-capability.<MODULE NAME>.<ACTION>` for a system capability and `namespace-capability.<MODULE_NAME>.<ACTION>` for a namespace capability.

1. List the capabilities of the module with the levels of the roles that collect them:

   ```
   d8 k get clusterroles -l rbac.deckhouse.io/kind=capability,module=<MODULE_NAME> -o json | jq -r '
     .items[]
     | [.metadata.labels["rbac.deckhouse.io/capability"],
        (.metadata.labels | to_entries
          | map(select(.key | test("^rbac\\.deckhouse\\.io/aggregate-to-.+-as$"))
   | (.key | ltrimstr("rbac.deckhouse.io/aggregate-to-") | rtrimstr("-as")) + "="
   + .value)
          | join(","))]
     | @tsv'
   ```

2. Keep the capabilities that the old selector matched. A selector with the `<LEVEL>` value matched the capabilities of this level and, through the built-in roles of the lower levels, the capabilities of those levels as well.
3. In the new role, put one selector by the `rbac.deckhouse.io/capability` label for each of them. To give all the system capabilities of the module whatever their level, a single selector by `rbac.deckhouse.io/scope: system` and `module: <MODULE_NAME>` is enough.

For example, the following namespace role gives in a namespace what a selector by `rbac.deckhouse.io/aggregate-to-virtualization-as: user` gave, without the other modules of the subsystem. Replace `<ACTION>` with the values from the list:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:namespace:virtualization-user
  labels:
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: namespace
    rbac.deckhouse.io/delegatable: "true"
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/capability: namespace-capability.virtualization.<ACTION>
    - matchLabels:
        rbac.deckhouse.io/capability: namespace-capability.virtualization.<ACTION>
rules: []
```

A role of the system or subsystem scope selects the `system-capability.<MODULE_NAME>.<ACTION>` values the same way.

Replacing a capability

A custom capability of the old scheme carries the `rbac.deckhouse.io/kind: manage` or `use` label and an aggregation label, and it is listed in the `D8UserAuthzForeignAggregationLabel` alert. While it carries the aggregation label of the system scope, of the namespace or project scope, or of a built-in subsystem, the `rbacv2-cluster-roles.deckhouse.io` webhook refuses every update of it. The webhook admits such a label only together with `rbac.deckhouse.io/kind: custom-capability`, and

that label requires the `d8:custom:` name prefix, which an existing object cannot get. Create a capability of the new scheme with the same rules in its place, and delete the old one once the new one grants the access.

The labels of the old capability define the new one:

Old capability	New capability
`rbac.deckhouse.io/kind: use` + `rbac.deckhouse.io/aggregate-to-kubernetes-as: <LEVEL>`	`d8:custom:namespace-capability:<NAME>` with `rbac.deckhouse.io/scope: namespace` and `rbac.deckhouse.io/aggregate-to-namespace-as: <LEVEL>`
`rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-<SUBSYSTEM>-as: <LEVEL>`, for a built-in subsystem or a subsystem of your own	`d8:custom:subsystem-capability:<NAME>` with `rbac.deckhouse.io/scope: subsystem`, `rbac.deckhouse.io/subsystem: <SUBSYSTEM>` and the same aggregation label
`rbac.deckhouse.io/kind: use` + `rbac.deckhouse.io/aggregate-to-all-as: <LEVEL>`	`d8:custom:namespace-capability:<NAME>` with `rbac.deckhouse.io/scope: namespace` and `rbac.deckhouse.io/aggregate-to-namespace-as: <LEVEL>`
`rbac.deckhouse.io/kind: manage` + `rbac.deckhouse.io/aggregate-to-all-as: <LEVEL>`	`d8:custom:system-capability:<NAME>` with `rbac.deckhouse.io/scope: system` and `rbac.deckhouse.io/aggregate-to-system-as: <LEVEL>`

Each new capability also gets the `rbac.deckhouse.io/kind: custom-capability` label.

An old capability with the `rbac.deckhouse.io/namespace: <NAMESPACE>` label gave the subjects of the roles that aggregate it RoleBinding objects in that namespace, the namespace of a module. Deckhouse Platform reads this label only on the ClusterRoles it ships, so neither the old capability nor its custom replacement gives such RoleBinding objects. To keep the access in that namespace, bind the subjects to a namespace role there with a RoleBinding (for details, see Extending subsystem roles and adding a new namespace).

A capability with the `rbac.deckhouse.io/kind: use` and `rbac.deckhouse.io/aggregate-to-kubernetes-as: <LEVEL>` labels extended the `d8:use:role:<LEVEL>` roles. Only the deprecated `d8:use:role:<LEVEL>` aliases still select it, until they are removed. `d8:namespace:<LEVEL>` and `d8:project:<LEVEL>` select `rbac.deckhouse.io/aggregate-to-namespace-as`, so a RoleBinding moved to a new role does not get the rules of the capability. Such a capability is listed in the `D8UserAuthzLegacyUseCapability` alert. Replace it before you move the RoleBinding objects to the new roles. For example, the following commands create its counterpart for the namespace roles:

```
OLD_CAPABILITY=<OLD_CAPABILITY_NAME>
NEW_CAPABILITY=d8:custom:namespace-capability:<NAME>
LEVEL=<LEVEL>
d8 k get clusterrole "$OLD_CAPABILITY" -o json | jq --arg name "$NEW_CAPABILITY" --arg level
"$LEVEL" '
  {apiVersion, kind, metadata: {name: $name, labels: {
     "rbac.deckhouse.io/kind": "custom-capability",
     "rbac.deckhouse.io/scope": "namespace",
     "rbac.deckhouse.io/aggregate-to-namespace-as": $level}},
   rules}' > new-capability.json
d8 k apply -f new-capability.json
```

Check the permissions of the subjects of the namespace roles, and then delete the old capability:

```
d8 k delete clusterrole "$OLD_CAPABILITY"
```

The `d8:use:role:<LEVEL>` aliases collect the new capability as well, so deleting the old one takes nothing away from their subjects.

Roles frozen earlier

A role frozen before the built-in capabilities got the new labels, by removing its `aggregationRule` field and keeping its rules, keeps its permissions and is not listed in the `D8UserAuthzLegacyRBACv2CustomRoleFound` alert.

- The explicit RoleBinding objects that replaced its automatic ones point at `d8:use:role:<LEVEL>`, which the new scheme keeps only as a deprecated alias of `d8:namespace:<LEVEL>`. They are listed in the `D8UserAuthzDeprecatedRBACv2RoleInUse` alert. Move them to `d8:namespace:<LEVEL>`, because a binding to an alias grants nothing once the alias is removed. The following commands do this for the RoleBinding objects with the `explicit:` name prefix:

  ```
  d8 k get rolebindings -A -o json | jq '{apiVersion: "v1", kind: "List", items:
  [.items[]
    | select(.roleRef.kind == "ClusterRole" and (.roleRef.name |
  startswith("d8:use:role:")) and (.metadata.name | startswith("explicit:")))]}' \
    > use-role-rolebindings.json
  jq '.items |= map({apiVersion, kind,
      metadata: {name: ("namespace:" + .metadata.name), namespace:
  .metadata.namespace,
        annotations: ((.metadata.annotations // {}) | with_entries(select(.key ==
  "rbac.deckhouse.io/related-with")))},
      roleRef: (.roleRef | .name |= (sub("^d8:use:role:"; "d8:namespace:") |
  sub(":kubernetes$"; "")))), subjects})' \
    use-role-rolebindings.json > namespace-rolebindings.json
  d8 k apply -f namespace-rolebindings.json && d8 k get -f namespace-rolebindings.json > /dev/null
  && d8 k delete -f use-role-rolebindings.json
  ```

  The old objects are deleted only when all the new ones exist, and the commands can be run again after a failure.

- If the role still carries the `rbac.deckhouse.io/aggregate-to-all-as` label, it is listed in the `D8UserAuthzForeignAggregationLabel` alert, and the `rbacv2-cluster-roles.deckhouse.io` webhook refuses every update of the role that keeps the label, a new apply of its manifest included. No role of the new model selects this label. The deprecated `d8:manage:all:<LEVEL>` alias still selects it on a role with `rbac.deckhouse.io/kind: manage`, and the deprecated `d8:use:role:<LEVEL>` alias on a role with `rbac.deckhouse.io/kind: use`, until the aliases are removed.

  With `rbac.deckhouse.io/kind: manage`, the subjects of `d8:manage:all:<LEVEL>` keep the role through the ClusterRoleBinding objects that the freeze created for them, so remove the label:

  ```
  d8 k label clusterrole <ROLE_NAME> rbac.deckhouse.io/aggregate-to-all-as-
  ```

  With `rbac.deckhouse.io/kind: use`, the role gives its rules to the subjects of every RoleBinding to `d8:use:role:<LEVEL>` through this label, and the freeze created no binding for them. Removing the label takes these rules away from them. First create a namespace capability with the rules of the role as described in Replacing a capability. Both `d8:namespace:<LEVEL>` and the `d8:use:role:<LEVEL>` alias collect it. Check the permissions of the subjects, and only then remove the label from the role or delete the role.

- A frozen role does not gain permissions for resources added later, such as the resources of a new module. To get them, replace it with a role of the new scheme as described in Fixing a role.

The RoleBindings that the previous release created in the module namespaces for a binding to a custom role with `rbac.deckhouse.io/kind: manage` and `rbac.deckhouse.io/use-role` are removed on the upgrade. Once the new role carries `rbac.deckhouse.io/scope: system` or `subsystem` and `rbac.deckhouse.io/use-role` and the binding refers to it, they come back as `d8:namespace:<level>:binding:<ClusterRoleBinding name>`, but only in the namespaces of the modules that already carry the new labels and whose system capabilities the new role aggregates. A module that still carries the old labels, such as `rbac.deckhouse.io/kind: manage`, gets no such RoleBinding until it moves to the new ones. Until then, create RoleBinding objects in the namespace of that module for the subjects of the binding: one to `d8:namespace:<level>` and one to each ClusterRole of the module that is listed in the `D8UserAuthzLegacyUseCapability` alert. None of these roles is an alias, so the RoleBinding objects do not hold the next release, and the webhook that checks RoleBinding objects in project namespaces does not check the namespace of a module. Delete them once the module carries the new labels and the controller creates its own RoleBinding there. A RoleBinding to `d8:use:role:<level>` gives these rights as well, but the `D8UserAuthzDeprecatedRBACv2RoleInUse` alert reports it, and it holds the release that removes the aliases.

Examples

Custom role before and after

The following is a configuration example of a role combining the permissions of the `deckhouse` and `kubernetes` subsystems and the `user-authn` module.

- Before (old scheme):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
rules: []
```

- After (new scheme):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: viewer
    rbac.deckhouse.io/kind: custom-role
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/aggregate-to-cluster-as: manager
    - matchLabels:
        rbac.deckhouse.io/scope: system
        module: user-authn
rules: []
```

What changed:

- The name got the mandatory `d8:custom:` prefix and names the subsystem of the role
- `rbac.deckhouse.io/kind: manage` → `rbac.deckhouse.io/kind: custom-role`
- `rbac.deckhouse.io/level: subsystem` → `rbac.deckhouse.io/scope: subsystem`
- The `rbac.deckhouse.io/aggregate-to-all-as` label is removed. The subjects of `d8:system:manager` get the role through a ClusterRoleBinding (for details, see Roles aggregated into the system roles)
- The `rbac.deckhouse.io/kind: manage` label is removed from the aggregation selectors
- All system permissions of a module are now selected with `rbac.deckhouse.io/scope: system` + `module: <module>`
- The `deckhouse` and `kubernetes` subsystems are part of the `cluster` subsystem now, so their two selectors became one. It selects the whole `cluster` subsystem, which is wider than the two old ones. To keep the role as narrow as before, select the system capabilities of the modules you need by `module: <module>` and `rbac.deckhouse.io/scope: system` instead

Custom capability before and after

The following is an example configuration of a capability, which grants read access to the MySuperResource resource and is aggregated into the role from the example above (its `aggregationRule` field must contain the `rbac.deckhouse.io/aggregate-to-mycustom-as: manager` selector).

- Before (old scheme):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:permission:mycustom:superresource:view
  labels:
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/aggregate-to-mycustom-as: manager
rules:
  - apiGroups:
      - mygroup.io
    resources:
      - mysuperresources
    verbs:
      - get
      - list
      - watch
```

- After (new scheme):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: d8:custom:subsystem-capability:mycustom:superresource:view
  labels:
    rbac.deckhouse.io/kind: custom-capability
    rbac.deckhouse.io/scope: subsystem
    rbac.deckhouse.io/subsystem: mycustom
    rbac.deckhouse.io/aggregate-to-mycustom-as: manager
rules:
  - apiGroups:
      - mygroup.io
    resources:
      - mysuperresources
    verbs:
      - get
      - list
      - watch
```

Labels and annotations: before and after

Labels on ClusterRole objects:

Label	Before	After	Purpose
`rbac.deckhouse.io/kind`	`manage` or `use`	`custom-role` / `custom-capability` for your own objects; `role` / `capability` on built-in ones (reserved)	The object type in the role model. Mandatory, objects without it are not processed
`rbac.deckhouse.io/level`	`all` | `subsystem` | `module`	Removed	The old role level; replaced by the `scope` label

Label	Before	After	Purpose
`rbac.deckhouse.io/scope`	—	`system` | `subsystem` | `namespace`	The scope of a role or capability
`rbac.deckhouse.io/subsystem`	Subsystem name	Unchanged	The role's subsystem; used with `scope: subsystem`
`rbac.deckhouse.io/use-role`	A use-role level	A namespace-role level	Defines which namespace role is automatically granted to the holder of a system/subsystem role in the system namespaces of its modules (via automatically created RoleBinding objects); `viewer` at most for a custom role
`rbac.deckhouse.io/aggregate-to-all-as`	`<level>`	With `rbac.deckhouse.io/kind: manage`, replaced by `rbac.deckhouse.io/aggregate-to-system-as` on a custom capability, which a custom role cannot carry. With `rbac.deckhouse.io/kind: use`, replaced by `rbac.deckhouse.io/aggregate-to-namespace-as` on a custom capability	Aggregates the object into the system-wide role (`d8:system:<level>`)
`rbac.deckhouse.io/aggregate-to-<subsystem>-as`	Used in selectors together with `rbac.deckhouse.io/kind: manage`	Used in selectors on its own	Aggregates the object into the subsystem role of the given level
`rbac.deckhouse.io/aggregate-to-kubernetes-as`	`<level>` (for use and manage permissions)	Not used. The kubernetes subsystem is part of the `cluster` subsystem (`aggregate-to-cluster-as`), and the old *use*-role meaning moved to `aggregate-to-namespace-as`	Aggregated into the old `d8:manage:kubernetes:<level>` and `d8:use:role:<level>:kubernetes` roles
`rbac.deckhouse.io/namespace`	Namespace	Unchanged	An additional namespace where a RoleBinding is automatically created for the role holders; read only on the capabilities Deckhouse ships, ignored on a custom one
`rbac.deckhouse.io/capability`	—	A unique capability name (for example, `system-capability.deckhouse.view`)	A machine-readable identifier of a built-in capability
`rbac.deckhouse.io/deprecated`	—	`"true"` on alias roles	The role is deprecated and will be removed; migrate the bindings to the new role

Label	Before	After	Purpose
`module`	Module name	Unchanged	Marks a built-in object as belonging to a Deckhouse Platform module; handy in aggregation selectors together with `scope`
`heritage: deckhouse`	Platform object marker	Unchanged	Must not be set on custom objects

Annotations on ClusterRole objects (the old scheme did not use annotations):

Annotation	Purpose
`ru.meta.deckhouse.io/title`, `ru.meta.deckhouse.io/description`	The displayed name and description of a role/capability in Russian (the platform sets them on built-in objects; you can set your own on custom ones)
`en.meta.deckhouse.io/title`, `en.meta.deckhouse.io/description`	Same in English
`rbac.deckhouse.io/deprecated-replaced-by`	Set on the alias roles of the previous names and names the new role to move the bindings to. An alias aggregates the capabilities of the **new** role, so the existing bindings to the previous name keep granting access until the aliases are removed. The permissions are not identical to the previous ones, because `d8:use:role:admin` no longer grants ServiceAccount token minting or impersonation

Adding a custom capability (in the new scheme)

A capability is a regular ClusterRole object with rules that is automatically included into the chosen role via an aggregation label. In the new scheme, a custom capability is created as follows:

1. Decide which role you want to extend: a namespace role, a subsystem role, the system role, or your own custom role.
2. Create a ClusterRole named `d8:custom:<SCOPE>-capability:<NAME>` with the `rbac.deckhouse.io/kind: custom-capability` label, the `rbac.deckhouse.io/scope: <SCOPE>` label, and the aggregation label of the target role:
   - `rbac.deckhouse.io/aggregate-to-namespace-as: <viewer|user|manager|admin|superadmin>`: Into the `d8:namespace:<level>` namespace role.
   - `rbac.deckhouse.io/aggregate-to-<subsystem>-as: <viewer|manager|superadmin>`: Into the `d8:subsystem:<subsystem>:<level>` subsystem role.
   - `rbac.deckhouse.io/aggregate-to-system-as: <viewer|manager|superadmin>`: Into the `d8:system:<level>` system role.
   - `rbac.deckhouse.io/aggregate-to-<your subsystem name>-as: <level>`: Into your own custom role (its `aggregationRule` field must contain such a selector).
3. Define the permissions in `rules`.

Kubernetes aggregates the rules automatically: right after the capability is created, its permissions appear for all holders of the target role. You can verify the result with `d8 k auth can-i --as <user>` or by inspecting the resulting role rules: `d8 k get clusterrole <role> -o yaml`.

For configuration examples, refer to the “Custom role before and after” and “Custom capability before and after” subsections.

How do I rename a built-in role?

The permissions of built-in roles cannot be changed, but their display title and description can — for example, so that the interface shows them in your company’s terms. Add the `custom.meta.deckhouse.io/title` and `custom.meta.deckhouse.io/description` annotations to the role:

```
d8 k annotate clusterrole d8:namespace:admin custom.meta.deckhouse.io/title='Team administrator'
custom.meta.deckhouse.io/description='Manages resources and access in the team namespace'
```

This is the only modification allowed for objects with the `d8:` prefix (except `d8:custom:*`): an attempt to change the rules, aggregation, or labels of a built-in role is rejected.

How do I find out who has access to a resource?

With the multitenancy mode enabled (`enableMultiTenancy`), a reverse authorization query is available — the WhoCan resource. It answers the question “who can perform action X on resource Y?” and returns the list of users, groups, and ServiceAccounts:

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: WhoCan
metadata:
  name: who-can-create-networkpolicies
spec:
  resourceAttributes:
    namespace: my-namespace
    verb: create
    group: networking.k8s.io
    resource: networkpolicies
EOF
```

The answer is returned in the `status` field (`users`, `groups`, `serviceAccounts`) directly in the command output; the object is not stored anywhere.

The right to create WhoCan queries is granted by the `d8:user-authz:who-can-checker` cluster role. It is not bound to anyone by default: the query result discloses access subjects across all namespaces, so grant it only to trusted administrators via a ClusterRoleBinding.

The managers of the `iam` and `security` subsystems have this right as well. It comes with the `d8:subsystem-capability:iam:inspect_access` capability of `d8:subsystem:iam:manager` and `d8:subsystem:security:manager`, and so the roles above them have it too (`d8:subsystem:iam:superadmin`, `d8:subsystem:security:superadmin`, `d8:system:manager` and `d8:system:superadmin`).

How do I find out what a specific user, group, or ServiceAccount is allowed to do?

With the multitenancy mode enabled (`enableMultiTenancy`), the SubjectAccessReport resource is available — the counterpart of WhoCan. It answers the question “what is this subject allowed to do” and returns a ready-made report: which roles are granted through which bindings, which actions on which resources are allowed cluster-wide and in every namespace, and where each permission comes from.

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: SubjectAccessReport
metadata:
  name: what-can-jane-do
spec:
  subject:
    kind: User
    name: jane@example.com
EOF
```

The `spec.subject.kind` field accepts `User`, `Group`, and `ServiceAccount` (for the latter, `spec.subject.namespace` is required). If `spec.subject` is omitted, the report is built for the caller.

Report specifics:

- The user’s groups are resolved automatically from the Group catalog, including nested ones: if the user belongs to group `B` and `B` belongs to `A`, the permissions of `A` are included as well. The groups taken into account are returned in `status.subject.groups`. Groups that are not in the catalog (for example, the ones coming from an external identity provider) can be passed in `spec.groups`.
- Every source of a permission (`status.scopes[].resources[].sources[]`) carries a `matchedBy` field showing whether the permission was granted to the subject personally or through a group, which makes it possible to view the access with group-derived grants excluded.
- Namespaces with identical access are merged into a single section

(`status.scopes[].namespaces`), so a project with a dozen namespaces does not turn into a dozen identical tables.
- Permissions granted by a ClusterRoleBinding are reported once, in the cluster-wide scope (`status.scopes[].cluster: true`), since they apply in every namespace.
- The report can be limited to specific namespaces with `spec.namespaces`.

The report is built from RBAC data. Admission webhooks can still refuse a write that RBAC allows, for example a change of an object labeled `heritage: deckhouse`, and the report does not show such refusals. If the subject's permissions are limited by a ClusterAuthorizationRule, this is reported in `status.notes`.

The right to build a report about **another** subject is granted by the `d8:user-authz:subject-access-checker` cluster role. Like `who-can-checker`, it is intentionally not bound to anyone by default: the report discloses the full permission map of the subject, including other namespaces. A report about oneself is available to every authenticated user and requires no extra permissions.

The managers of the `iam` and `security` subsystems have this right as well. It comes with the `d8:subsystem-capability:iam:inspect_access` capability of `d8:subsystem:iam:manager` and `d8:subsystem:security:manager`, and so the roles above them have it too (`d8:subsystem:iam:superadmin`, `d8:subsystem:security:superadmin`, `d8:system:manager` and `d8:system:superadmin`).

How does a user see the list of namespaces available to them?

With the multitenancy mode enabled (`enableMultiTenancy`), the namespace list is filtered automatically: the `d8 k get namespaces` command returns to a user only the namespaces they have access to — via any of the mechanisms (role bindings, ProjectRoleBinding/ClusterProjectRoleBinding, ClusterAuthorizationRule/AuthorizationRule). A user does not see foreign namespaces and cannot learn about their existence from the list.

If the `console` module older than 1.44.0 is enabled, the list is not filtered: the access levels of ClusterAuthorizationRule and AuthorizationRule and the namespace roles keep the permission to list every namespace.

The same list is served by the read-only `accessiblenamespaces` resource — any authenticated user can query it **for themselves**:

```
d8 k get accessiblenamespaces
```

This is convenient for scripts and interfaces: there is no need to iterate over namespaces checking access to each one.

How does an interface find out where a user may perform an action?

With the multitenancy mode enabled (`enableMultiTenancy`), the SelfScopedAccessReview resource gives the user who creates it with information about the namespaces in which they are allowed to perform actions within the filtering scope: in the whole cluster, in how many namespaces of the scope and, on request, in which ones. One query replaces an access check per namespace. The namespaces are counted by the rule that filters a list sent with the same scope, so a section built on the answer opens exactly when its list has objects to return.

Any authenticated user can create the resource, and it answers only about the caller. A query with the `spec.user`, `spec.groups`, `spec.uid` or `spec.extra` fields is refused.

Example query from a Deckhouse web interface page that shows deployments of all projects, offers to create one and shows pod logs:

```
d8 k create -o yaml -f - <<EOF
apiVersion: authorization.deckhouse.io/v1alpha1
kind: SelfScopedAccessReview
spec:
  scope: projects
  requests:
  - group: apps
    resource: deployments
    verbs: [list, watch]
  - group: apps
    resource: deployments
    verbs: [create]
    includeNamespaces: true
  - group: ""
    resource: pods
    subresource: log
    verbs: [get]
    includeNamespaces: true
EOF
```

Example answer for a user who administers the `shop-dev` and `shop-stage` namespaces of a project and has read access without pod logs in `shop-prod`:

```
status:
  results:
  - allowed: true
    cluster: false
    count: 3
  - allowed: true
    cluster: false
    count: 2
    namespaces: [shop-dev, shop-stage]
  - allowed: true
    cluster: false
    count: 2
    namespaces: [shop-dev, shop-stage]
```

Example answer for a cluster administrator. The namespaces are neither counted nor listed, even with `includeNamespaces: true`:

```
status:
  results:
  - allowed: true
    cluster: true
  - allowed: true
    cluster: true
  - allowed: true
    cluster: true
```

Example answer for a user who can only read deployments in the `billing-prod` namespace of another project:

```
status:
  results:
  - allowed: true
    cluster: false
    count: 1
  - allowed: false
    cluster: false
    count: 0
    namespaces: []
  - allowed: false
    cluster: false
    count: 0
    namespaces: []
```

The query fields:

- `spec.scope`: Scope of the `X-Deckhouse-Scope` header (filtering scope): `accessible`, `projects`, `system` or `project`.
- `spec.project`: Project of the `project` scope.
- `spec.requests`: Actions to check. Each sets the API group (`group`, empty for the core group), the resource (`resource`), the subresource if any (`subresource`) and the verbs (`verbs`). `includeNamespaces: true` asks for the names of the namespaces.

The `status.results` list has an entry per request, in the same order as the requests in the SelfScopedAccessReview:

- `allowed`: The action is allowed in the whole cluster or in at least one namespace of the scope.
- `cluster`: The action is allowed in the whole cluster. The namespaces are then neither counted nor listed.
- `count`: Number of namespaces of the scope where the action is allowed.
- `namespaces`: Namespaces of the scope where the action is allowed, sorted alphabetically, when the request sets `includeNamespaces: true`. An empty list means none.
- `reason`: `NotFilterable` when the action on a namespaced resource cannot be answered by namespaces: the list of the resource is not filtered by scope. Only a permission in the whole cluster allows such an action.
- `evaluationError`: Reason why this request could not be checked. The other requests are answered.

The namespaces where the action is allowed are counted as follows:

- A namespace counts when every verb of the request is allowed in it. A request for `list` and `watch` counts the namespaces where both are allowed.
- Only the namespaces the user sees in the `accessiblenamespaces` list are counted and named, and the namespaces where a RoleBinding grants access to every authenticated user (the `system:authenticated` group): RBAC opens them to the user, and the filtered list returns their objects.
- For `get`, `list` and `watch` of a resource, the answer names the namespaces the list with the same scope returns objects from.
- Other verbs and subresources, for example `create` or `pods/log`, are checked in every namespace of the scope.
- `get`, `list` and `watch` of `namespaces` are answered with the accessible namespaces whatever the scope, as the list of namespaces is filtered.
- `get`, `list` and `watch` of `deckhouse.io/projects` are answered with every accessible namespace: each stands for its project, and a namespace without the `projects.deckhouse.io/project` label for the virtual project `multitenancy-manager` counts it in, `deckhouse` for `d8-*`, `kube-*`, `upmeter-*` and namespaces with the `heritage: deckhouse` or `heritage: upmeter` label, `default` for the rest. So `count` and `namespaces` list namespaces, not projects: a project with three accessible namespaces counts three. A user who may list namespaces in the whole cluster sees every project and gets `cluster: true`.
- `update`, `patch` and `delete` of `namespaces` and their subresources are checked for every namespace of the scope, with that namespace as the target: the `ClusterAuthorizationRule` limits apply to them.
- When a request for `namespaces` combines a read with a write, a namespace opened to every authenticated user but missing from `accessiblenamespaces` counts only if `get` is allowed in it and `list` and `watch` in the whole cluster, as the API server checks them.
- `create` and `deletecollection` of `namespaces`, every write of `deckhouse.io/projects`, and every action on another cluster-scoped resource, such as `nodes` or `ModuleConfig`, are allowed only by a permission in the whole cluster. Without one the answer is `allowed: false`, with no `count` and no `reason`.
- A read of a namespaced resource the scope does not filter gets `reason: NotFilterable` unless it

is allowed in the whole cluster. These are resources without `list`, resources of aggregated APIs such as `metrics.k8s.io`, and resources the cluster does not serve, `deckhouse.io/projects` without the `multitenancy-manager` module included: their lists with a scope are refused.
- In the `projects` and `project` scopes, a namespace with an empty `projects.deckhouse.io/project` label belongs to no project. The answer for `deckhouse.io/projects` counts it for its virtual project, as described above.
- The `accessible` and `system` scopes include the `default`, `kube-*` and `d8-*` namespaces, as their lists do. The `system` and `projects` scopes can overlap: the namespaces of a project named `d8`, such as `d8-x`, belong to both.

The answer describes authorization, the decision whether the user may perform the action on the resource. Admission can still refuse a write of a particular object:

- creation, update and delete of objects with the `heritage: deckhouse` label, for everyone except platform components;
- update and delete of objects that `multitenancy-manager` creates in project namespaces, and deletion of a project namespace;
- creation of a namespace while the `allowNamespacesWithoutProjects` parameter of the `multitenancy-manager` module is `false`;
- creation of a ProjectRoleBinding or a ProjectNamespace outside the main namespace of the project;
- creation of users, groups and `ClusterAuthorizationRule` objects with rights beyond the user's own.

The answer cannot express a permission for named objects only (`resourceNames` in a role): it asks about a resource without an object name, so such a permission is not counted. A change of rules or bindings reaches the answer within seconds, while the API server can keep its previous decision for a repeated request for up to 30 seconds.

While the `ClusterAuthorizationRules` are not yet loaded or no longer tracked, the whole query is answered with `503` and the `Retry-After` header: the API server refuses requests in that state too, and an answer computed without the rules would read as a refusal.

The answer applies the `ClusterAuthorizationRule` limits whenever the authorization webhook of `user-authz` is enabled through the `controlPlaneConfigurator.enabled` parameter and the `control-plane-manager` module, which puts it into the API server configuration, is enabled. If a third-party authorization webhook is set by hand in the `apiserver.authz.webhookURL` parameter of the `control-plane-manager` module, the API server may not apply these limits, while the answer still does.

To keep an interface consistent with the data it shows, follow these rules:

1. Ask with the scope of the list the page shows.
2. Show a section when `allowed` is `true`.
3. Cache answers per scope and project.
4. Treat an empty list returned with a scope as the absence of objects, not as missing permissions.
5. Take the namespaces of a creation form from `namespaces` of a request with `includeNamespaces: true`.

The `projects` and `project` scopes require the `multitenancy-manager` module to be enabled in the cluster. Without it, a query with these scopes is refused with `400`, and the `accessible` and `system` scopes work as usual. The cluster then serves no projects, and requests for `deckhouse.io/projects` get `reason: NotFilterable` unless they are allowed in the whole cluster.

The server answers with an error in the following cases:

- `400`: The query is invalid. Examples are an unknown scope, more than 1000 requests, more than five requests with `includeNamespaces: true`, more than 16 verbs in one request, or more than 10,000,000 checks of a verb in a namespace. A user with 10,000 namespaces reaches the last bound with 1000 requests of one verb, or 500 requests of two: split such a query into several.
- `429`: The same user has 30 queries in progress.
- `504`: The query was not answered within 10 seconds.
- `503`: `permission-browser-apiserver` is unavailable, or it has not loaded the `ClusterAuthorizationRules` yet (with `Retry-After`).

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The vertical-pod-autoscaler module

> **i** The module lifecycle stage: General Availability
>
> The module has requirements for installation

Vertical Pod Autoscaler (VPA) is an infrastructure service that allows you to avoid setting exact resource requests for a container if their values are not known in advance. VPA can automatically adjust the CPU and memory reservations (providing that the corresponding mode is enabled) based on the actual resource consumption (as shown by the Prometheus data). Also, VPA can recommend values for resource requests and limits without updating them automatically.

VPA has the following operating modes:

- `InPlaceOrRecreate`: A default mode. Available in Kubernetes 1.33 and later. VPA attempts to update Pod resources without recreating the Pod. If in-place resource updates are not possible, VPA recreates the Pod similarly to the `Recreate` mode.
- `Recreate`: The mode allows VPA to change resources of running pods (restart them while running). In case of one pod running (`replicas: 1`), this will lead to the service being unavailable during the restart. In this mode, VPA does not recreate pods that were created without a controller.
- `Initial`: VPA modifies Pod resources only when Pods are started (but not during operation).
- `Off`: VPA does not take any action to update the resource requests for the running containers. In this case, if VPA is running in this mode, you can see what resource values it recommends (`d8 k describe vpa <vpa-name>`).

> **⚠** The `Auto` mode is deprecated. It will no longer be supported in future DP versions. Switch to one of the supported operating modes.

VPA limitations:

- When using modes that allow Pod recreation, VPA may recreate a Pod if its `resource requests` cannot be updated in place. The new Pod can be scheduled to another node.
- VPA **should not be used concurrently with CPU and memory-based HPA**. However, you can use VPA together with HPA for custom/external metrics.
- VPA notices almost all `out-of-memory` events, but that does not guarantee its response.
- VPA performance has not been tested for huge clusters.
- VPA recommendations may exceed the available resources in the cluster. That can lead to **Pods becoming Pending**.
- Using multiple VPAs for the same Pod can lead to undefined behavior.
- If VPA is deleted or “turned off” (the `Off` mode), the changes made by VPA earlier are not reset (the most recent value set is kept). It may lead to confusion due to the difference between resource values in Helm/controller and the actual resources of Pods (it may be perceived as if they “came from out of nowhere”).

> **i** We highly recommend using Pod Disruption Budget with VPA.

Grafana dashboard

Displayed on dashboards:

- `Main / Namespace`, `Main / Namespace / Controller`, `Main / Namespace / Controller / Pod` — the `VPA type` column shows the value of `updatePolicy.updateMode`;
- `Main / Namespaces` — the `VPA %` column shows the percentage of VPA-enabled Pods.

Vertical Pod Autoscaler Architecture

VPA consists of 3 components:

- `Recommender` — this component monitors the current resource consumption (by making requests to the Metrics API implemented in the `prometheus-metrics-adapter` module) as well as consumption history (by making requests to Trickster caching proxy). As its name suggests, the component provides CPU and memory recommendations for containers.
- `Updater` — this component checks if the Pods have correct resources set and, if not, kills them so that they can be recreated by their controllers with the updated resource requests.
- `Admission Plugin` — this component sets the correct resource requests on new Pods (either just

created or recreated by their controller due to Updater's activity).

When the Updater component changes resource values, Pods are evicted using the Eviction API. Thus, the `Pod Disruption Budget` is taken into account for the Pods being updated.

Third-party components

List of third-party software used in the `vertical-pod-autoscaler` module:

- Kubernetes Autoscaler 1.8.1

  License: Apache License 2.0

  A autoscaling-related components for Kubernetes.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

The vertical-pod-autoscaler: configuration

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

VPA works directly with the Pod (instead of the Pod controller) by measuring and changing its containers' parameters. Configuring is performed using the `VerticalPodAutoscaler` Custom Resource.

The module generally requires no configuration and only has the `nodeSelector/tolerations` and `logLevel` settings.

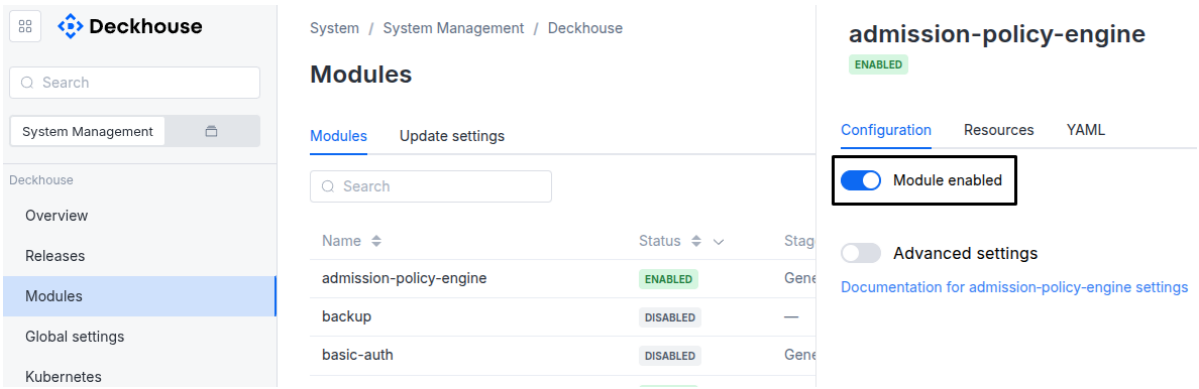
The module has 1 alert.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `vertical-pod-autoscaler` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable vertical-pod-autoscaler
```

- **Using ModuleConfig `vertical-pod-autoscaler`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `vertical-pod-autoscaler` (create it if necessary);

  Example of a manifest to enable module `vertical-pod-autoscaler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  enabled: true
```

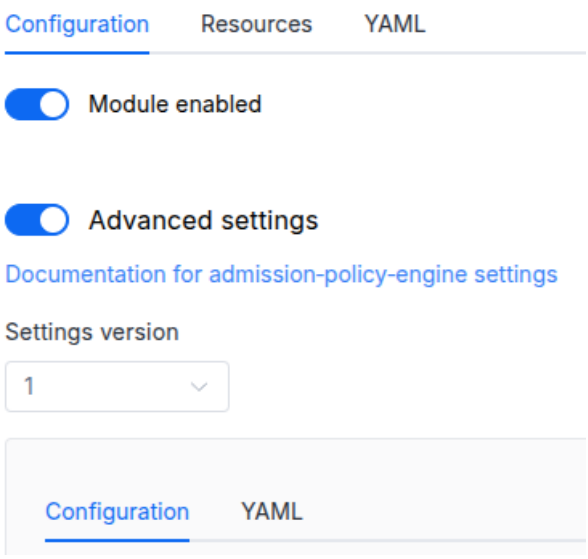
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `vertical-pod-autoscaler` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `vertical-pod-autoscaler` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `vertical-pod-autoscaler`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `vertical-pod-autoscaler` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc vertical-pod-autoscaler
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `vertical-pod-autoscaler` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `vertical-pod-autoscaler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Requirements

To the versions of other modules:

- `prometheus`: any version.
- `prometheus-metrics-adapter`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.logLevel**
    string

    Logging level of the VPA components: recommender, updater, and admission-controller.

    Each level includes the messages of the less detailed levels. Some informational messages and warnings, such as startup messages, are logged at any level.

    Supported values:

    - `Error`: Errors and important messages that VPA logs at the info level, such as failed evictions and failed in-place updates.
    - `Warning`: Warnings, the component version at startup, and the pod evictions caused by node resource pressure that the recommender processes (default).
    - `Info`: Actions on pods, such as evictions, in-place resource updates, and setting the resources of pods being created.
    - `Debug`: The progress of each recommender cycle and detailed information on decisions for each pod and container, such as why a pod was not updated. Produces a very large volume of logs, so use it for troubleshooting only.
    - `Trace`: Every request to the Kubernetes API. Use it only when `Debug` is not enough to find the problem.

    Default: `Warning`

    Allowed values: `Trace`, `Debug`, `Info`, `Warning`, `Error`

  - **settings.nodeSelector**
    object

    The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations**
    array of objects

    The same as in the pods' `spec.tolerations` parameter in Kubernetes;

    If the parameter is omitted or `false`, it will be determined automatically.

    - **settings.tolerations.effect**
      string
    - **settings.tolerations.key**
      string
    - **settings.tolerations.operator**

string
- **settings.tolerations.tolerationSeconds**
integer
- **settings.tolerations.value**
string

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Custom resources

> The Deckhouse Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

VerticalPodAutoscaler

Short names: vpa

Scope: Namespaced

- v1
- v1beta2

Configuration for a vertical pod autoscaler, which automatically manages pod resources based on historical and real time resource utilization.

- **apiVersion**
string

  Defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values.

  More information in the documentation.

- **kind**
string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase.

  More information in the documentation.

- **metadata**
object
- **spec**
object

  Required value

  Specification of the behavior of the autoscaler.

  More information in the documentation.

  - **spec.recommenders**
  array of objects

    Recommender responsible for generating recommendation for this object. List should be empty (then the default recommender will generate the recommendation) or contain exactly one recommender.

    VerticalPodAutoscalerRecommenderSelector points to a specific Vertical Pod Autoscaler

recommender. In the future it might pass parameters to the recommender.

- **spec.recommenders.name**
  string

  Required value

  Name of the recommender responsible for generating recommendation for this object.

- **spec.resourcePolicy**
  object

  Controls how the autoscaler computes recommended resources. The resource policy may be used to set constraints on the recommendations for individual containers.

  If any individual containers need to be excluded from getting the VPA recommendations, then it must be disabled explicitly by setting mode to `Off` under `containerPolicies`.

  If not specified, the autoscaler computes recommended resources for all containers in the pod, without additional constraints.

  - **spec.resourcePolicy.containerPolicies**
    array of objects

    Per-container resource policies.

    Controls how autoscaler computes the recommended resources for a specific container.

    - **spec.resourcePolicy.containerPolicies.containerName**
      string

      Name of the container or `DefaultContainerResourcePolicy`, in which case the policy is used by the containers that don’t have their own policy specified.

    - **spec.resourcePolicy.containerPolicies.controlledResources**
      array of strings

      Specifies the type of recommendations that will be computed (and possibly applied) by VPA. Only `cpu` and `memory` are supported. If not specified, the default of (`cpu`, `memory`) will be used.

      Default: `["cpu","memory"]`

      - **Element of the array**
        string

        ResourceName is the name identifying various resources in a ResourceList.

    - **spec.resourcePolicy.containerPolicies.controlledValues**
      string

      Specifies which resource values should be controlled.

      Default: `RequestsAndLimits`

      Allowed values: `RequestsAndLimits`, `RequestsOnly`

    - **spec.resourcePolicy.containerPolicies.maxAllowed**
      object

      Specifies the maximum amount of resources that will be recommended for the container. The default is no maximum.

    - **spec.resourcePolicy.containerPolicies.minAllowed**
      object

      Specifies the minimal amount of resources that will be recommended for the container. The default is no minimum.

    - **spec.resourcePolicy.containerPolicies.mode**
      string

      Whether autoscaler is enabled for the container.

      Default: `Auto`

      Allowed values: `Auto`, `Off`

    - **spec.resourcePolicy.containerPolicies.recommendationFactor**

object

Per-resource multiplier for the recommendation computed from usage. It is applied after the recommender's own safety margin and before `minAllowed` and `maxAllowed`. `1.2` increases the recommendation by 20%, `0.87` decreases it by 13%. The result never goes below the module-wide minimum recommendation.

Keys are `cpu` and `memory`. Values are quantities written as strings with at most two decimal places, for example `"1.25"`. The allowed range is greater than zero and up to `10` inclusive. A missing key means `1`.

The multiplier affects `target`, `lowerBound` and `upperBound`. `uncappedTarget` keeps the value computed without the multiplier. Limits follow requests proportionally, as usual.

- **spec.scope**
  string

  Scope controls additional recommendation partitioning for DaemonSet targets.

  The value must be a node label key (for example: `node.kubernetes.io/instance-type`).

- **spec.targetRef**
  object

  Required value

  Points to the controller managing the set of pods for the autoscaler to control — e.g. Deployment, StatefulSet.

  VerticalPodAutoscaler can be targeted at controller implementing scale subresource (the pod set is retrieved from the controller's `ScaleStatus`) or some well known controllers (e.g. for DaemonSet the pod set is read from the controller's spec).

  If VerticalPodAutoscaler cannot use specified target it will report `ConfigUnsupported` condition. Note that VerticalPodAutoscaler does not require full implementation of scale subresource — it will not use it to modify the replica count. The only thing retrieved is a label selector matching pods grouped by the target resource.

  - **spec.targetRef.apiVersion**
    string

    API version of the referent.

  - **spec.targetRef.kind**
    string

    Required value

    Kind of the referent.

    More info.

  - **spec.targetRef.name**
    string

    Required value

    Name of the referent.

    More info.

- **spec.updatePolicy**
  object

  Describes the rules on how changes are applied to the pods. If not specified, all fields in the `PodUpdatePolicy` are set to their default values.

  - **spec.updatePolicy.evictionRequirements**
    array of objects

    EvictionRequirements is a list of EvictionRequirements that need to evaluate to true in order for a Pod to be evicted. If more than one EvictionRequirement is specified, all of them need to be fulfilled to allow eviction.

    EvictionRequirement defines a single condition which needs to be true in order to evict a Pod.

- **spec.updatePolicy.evictionRequirements.changeRequirement**
  string

  Required value

  EvictionChangeRequirement refers to the relationship between the new target recommendation for a Pod and its current requests, what kind of change is necessary for the Pod to be evicted.

  Allowed values: `TargetHigherThanRequests`, `TargetLowerThanRequests`

- **spec.updatePolicy.evictionRequirements.resources**
  array of strings

  Required value

  Resources is a list of one or more resources that the condition applies to. If more than one resource is given, the EvictionRequirement is fulfilled if at least one resource meets `changeRequirement`.

  - **Element of the array**
    string

    ResourceName is the name identifying various resources in a ResourceList.

- **spec.updatePolicy.minReplicas**
  integer

  Minimal number of replicas which need to be alive for Updater to attempt pod eviction (pending other checks like PDB). Only positive values are allowed. Overrides global '–min-replicas' flag.

- **spec.updatePolicy.updateMode**
  string

  The operation mode of the VPA controller.

  - `InPlaceOrRecreate`: A default mode. Available in Kubernetes 1.33 and later. VPA attempts to update Pod resources without a restart; if that is not possible, it recreates the Pod.
  - `Recreate`: This mode allows VPA to modify resource requirements of the running pods (i.e., restart them during operation). This mode might result in temporary unavailability of the service (due to restart) if you have just one replica (`replicas: 1`). In this mode, VPA does not recreate pods that were not created by a controller.
  - `Initial`: VPA modifies pod resources only when pods are started (but not when pods are running).
  - `Auto`: A deprecated mode. It will no longer be supported in future DP versions. Use one of the supported operating modes instead.
  - `Off`: VPA does not take any action to update the resource requests for the running containers. Autoscaler calculates recommendations and stores them in the status field of the VPA object. You can browse VPA recommendations using the following command: `d8 k describe vpa <vpa-name>`.

  Default: `InPlaceOrRecreate`

  Allowed values: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **status**
  object

  Current information about the autoscaler.

  - **status.conditions**
    array of objects

    Set of conditions required for this autoscaler to scale its target, and indicates whether or not those conditions are met.

    Describes the state of a VerticalPodAutoscaler at a certain point.

    - **status.conditions.lastTransitionTime**
      string

      Last time the condition transitioned from one status to another.

    - **status.conditions.message**

string

A human-readable explanation containing details about the transition.

- **status.conditions.reason**
  string

  Reason for the condition’s last transition.

- **status.conditions.status**
  string

  Required value

  Status of the condition (`True`, `False`, Unknown).

- **status.conditions.type**
  string

  Required value

  Describes the current condition.

- **status.groups**
  array of objects

  Contains grouped recommendations by scope value when `spec.scope` is used.

  - **status.groups.containerRecommendations**
    array of objects

    Resources recommended by autoscaler for each container in this group.

    - **status.groups.containerRecommendations.containerName**
      string

      Name of the container.

    - **status.groups.containerRecommendations.lowerBound**
      object

      Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

    - **status.groups.containerRecommendations.scope**
      string

      Scope binds this recommendation to a specific scope value in form “ =".

    - **status.groups.containerRecommendations.target**
      object

      Required value

      Recommended amount of resources. Observes `ContainerResourcePolicy`.

    - **status.groups.containerRecommendations.uncappedTarget**
      object

      The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`, including `recommendationFactor`.

      May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

    - **status.groups.containerRecommendations.upperBound**
      object

      Maximum recommended amount of resources.

      Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

        - **status.groups.scopeValue**
          string

          Required value

          Scope value of `spec.scope` label key for this recommendation group.

    - **status.recommendation**
      object

      The most recently computed amount of resources recommended by the autoscaler for the controlled pods.

        - **status.recommendation.containerRecommendations**
          array of objects

          Resources recommended by the autoscaler for each container.

          Recommendation of resources computed by autoscaler for a specific container. Respects the container resource policy if present in the spec. In particular the recommendation is not produced for containers with `ContainerScalingMode` set to 'Off'.

            - **status.recommendation.containerRecommendations.containerName**
              string

              Name of the container.

            - **status.recommendation.containerRecommendations.lowerBound**
              object

              Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

            - **status.recommendation.containerRecommendations.scope**
              string

              Scope binds this recommendation to a specific scope value in form " =".

            - **status.recommendation.containerRecommendations.target**
              object

              Required value

              Recommended amount of resources. Observes `ContainerResourcePolicy`.

            - **status.recommendation.containerRecommendations.uncappedTarget**
              object

              The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`, including `recommendationFactor`. May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

            - **status.recommendation.containerRecommendations.upperBound**
              object

              Maximum recommended amount of resources. Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

Deprecated resource. Support for the resource might be removed in a later release.

Configuration for a vertical pod autoscaler, which automatically manages pod resources based on historical and real time resource utilization.

- **apiVersion**
  string

  Defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values.

More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase.

  More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Required value

  Specification of the behavior of the autoscaler.

  More information in the documentation.

  - **spec.recommenders**
    array of objects

    Recommender responsible for generating recommendation for this object. List should be empty (then the default recommender will generate the recommendation) or contain exactly one recommender.

    VerticalPodAutoscalerRecommenderSelector points to a specific Vertical Pod Autoscaler recommender. In the future it might pass parameters to the recommender.

    - **spec.recommenders.name**
      string

      Required value

      Name of the recommender responsible for generating recommendation for this object.

  - **spec.resourcePolicy**
    object

    Controls how the autoscaler computes recommended resources. The resource policy may be used to set constraints on the recommendations for individual containers.

    If any individual containers need to be excluded from getting the VPA recommendations, then it must be disabled explicitly by setting mode to `Off` under `containerPolicies`.

    If not specified, the autoscaler computes recommended resources for all containers in the pod, without additional constraints.

    - **spec.resourcePolicy.containerPolicies**
      array of objects

      Per-container resource policies.

      Controls how autoscaler computes the recommended resources for a specific container.

      - **spec.resourcePolicy.containerPolicies.containerName**
        string

        Name of the container or `DefaultContainerResourcePolicy`, in which case the policy is used by the containers that don’t have their own policy specified.

      - **spec.resourcePolicy.containerPolicies.controlledResources**
        array of strings

        Specifies the type of recommendations that will be computed (and possibly applied) by VPA. Only `cpu` and `memory` are supported. If not specified, the default of (`cpu`, `memory`) will be used.

        Default: `["cpu","memory"]`

        - **Element of the array**
          string

          ResourceName is the name identifying various resources in a ResourceList.

      - **spec.resourcePolicy.containerPolicies.controlledValues**

string

Specifies which resource values should be controlled.

Default: `RequestsAndLimits`

Allowed values: `RequestsAndLimits`, `RequestsOnly`

- **spec.resourcePolicy.containerPolicies.maxAllowed**
  object

  Specifies the maximum amount of resources that will be recommended for the container. The default is no maximum.

- **spec.resourcePolicy.containerPolicies.minAllowed**
  object

  Specifies the minimal amount of resources that will be recommended for the container. The default is no minimum.

- **spec.resourcePolicy.containerPolicies.mode**
  string

  Whether autoscaler is enabled for the container.

  Default: `Auto`

  Allowed values: `Auto`, `Off`

- **spec.resourcePolicy.containerPolicies.recommendationFactor**
  object

  Per-resource multiplier for the recommendation computed from usage. It is applied after the recommender's own safety margin and before `minAllowed` and `maxAllowed`. `1.2` increases the recommendation by 20%, `0.87` decreases it by 13%. The result never goes below the module-wide minimum recommendation.

  Keys are `cpu` and `memory`. Values are quantities written as strings with at most two decimal places, for example `"1.25"`. The allowed range is greater than zero and up to `10` inclusive. A missing key means `1`.

  The multiplier affects `target`, `lowerBound` and `upperBound`. `uncappedTarget` keeps the value computed without the multiplier. Limits follow requests proportionally, as usual.

- **spec.scope**
  string

  Scope controls additional recommendation partitioning for DaemonSet targets.

  The value must be a node label key (for example: `node.kubernetes.io/instance-type`).

- **spec.targetRef**
  object

  Required value

  Points to the controller managing the set of pods for the autoscaler to control — e.g. Deployment, StatefulSet.

  VerticalPodAutoscaler can be targeted at controller implementing scale subresource (the pod set is retrieved from the controller's `ScaleStatus`) or some well known controllers (e.g. for DaemonSet the pod set is read from the controller's spec).

  If VerticalPodAutoscaler cannot use specified target it will report `ConfigUnsupported` condition. Note that VerticalPodAutoscaler does not require full implementation of scale subresource — it will not use it to modify the replica count. The only thing retrieved is a label selector matching pods grouped by the target resource.

  - **spec.targetRef.apiVersion**
    string

    API version of the referent.

  - **spec.targetRef.kind**
    string

    Required value

Kind of the referent.

More info.

- **spec.targetRef.name**
  string

  Required value

  Name of the referent.

  More info.

- **spec.updatePolicy**
  object

  Describes the rules on how changes are applied to the pods. If not specified, all fields in the `PodUpdatePolicy` are set to their default values.

  - **spec.updatePolicy.evictionRequirements**
    array of objects

    EvictionRequirements is a list of EvictionRequirements that need to evaluate to true in order for a Pod to be evicted. If more than one EvictionRequirement is specified, all of them need to be fulfilled to allow eviction.

    EvictionRequirement defines a single condition which needs to be true in order to evict a Pod.

    - **spec.updatePolicy.evictionRequirements.changeRequirement**
      string

      Required value

      EvictionChangeRequirement refers to the relationship between the new target recommendation for a Pod and its current requests, what kind of change is necessary for the Pod to be evicted.

      Allowed values: `TargetHigherThanRequests`, `TargetLowerThanRequests`

    - **spec.updatePolicy.evictionRequirements.resources**
      array of strings

      Required value

      Resources is a list of one or more resources that the condition applies to. If more than one resource is given, the EvictionRequirement is fulfilled if at least one resource meets `changeRequirement`.

      - **Element of the array**
        string

        ResourceName is the name identifying various resources in a ResourceList.

  - **spec.updatePolicy.minReplicas**
    integer

    Minimal number of replicas which need to be alive for Updater to attempt pod eviction (pending other checks like PDB). Only positive values are allowed. Overrides global ‘–min-replicas’ flag.

  - **spec.updatePolicy.updateMode**
    string

    The operation mode of the VPA controller.

    - `InPlaceOrRecreate`: A default mode. Available in Kubernetes 1.33 and later. VPA attempts to update Pod resources without a restart; if that is not possible, it recreates the Pod.
    - `Recreate`: This mode allows VPA to modify resource requirements of the running pods (i.e., restart them during operation). This mode might result in temporary unavailability of the service (due to restart) if you have just one replica (`replicas: 1`). In this mode, VPA does not recreate pods that were not created by a controller.
    - `Initial`: VPA modifies pod resources only when pods are started (but not when pods are running).
    - `Auto`: A deprecated mode. It will no longer be supported in future DP versions. Use one of the supported operating modes instead.

        - `Off`: VPA does not take any action to update the resource requests for the running containers. Autoscaler calculates recommendations and stores them in the status field of the VPA object. You can browse VPA recommendations using the following command: `d8 k describe vpa <vpa-name>`.

      Default: `InPlaceOrRecreate`

      Allowed values: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **status**
  object

  Current information about the autoscaler.

  - **status.conditions**
    array of objects

    Set of conditions required for this autoscaler to scale its target, and indicates whether or not those conditions are met.

    Describes the state of a VerticalPodAutoscaler at a certain point.

    - **status.conditions.lastTransitionTime**
      string

      Last time the condition transitioned from one status to another.

    - **status.conditions.message**
      string

      A human-readable explanation containing details about the transition.

    - **status.conditions.reason**
      string

      Reason for the condition’s last transition.

    - **status.conditions.status**
      string

      Required value

      Status of the condition (`True`, `False`, Unknown).

    - **status.conditions.type**
      string

      Required value

      Describes the current condition.

  - **status.groups**
    array of objects

    Contains grouped recommendations by scope value when `spec.scope` is used.

    - **status.groups.containerRecommendations**
      array of objects

      Resources recommended by autoscaler for each container in this group.

      - **status.groups.containerRecommendations.containerName**
        string

        Name of the container.

      - **status.groups.containerRecommendations.lowerBound**
        object

        Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

      - **status.groups.containerRecommendations.scope**
        string

        Scope binds this recommendation to a specific scope value in form “ =".

- **status.groups.containerRecommendations.target**
  object

  Required value

  Recommended amount of resources. Observes `ContainerResourcePolicy`.

- **status.groups.containerRecommendations.uncappedTarget**
  object

  The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`, including `recommendationFactor`.

  May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

- **status.groups.containerRecommendations.upperBound**
  object

  Maximum recommended amount of resources.

  Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

- **status.groups.scopeValue**
  string

  Required value

  Scope value of `spec.scope` label key for this recommendation group.

- **status.recommendation**
  object

  The most recently computed amount of resources recommended by the autoscaler for the controlled pods.

  - **status.recommendation.containerRecommendations**
    array of objects

    Resources recommended by the autoscaler for each container.

    Recommendation of resources computed by autoscaler for a specific container. Respects the container resource policy if present in the spec. In particular the recommendation is not produced for containers with `ContainerScalingMode` set to ‘Off’.

    - **status.recommendation.containerRecommendations.containerName**
      string

      Name of the container.

    - **status.recommendation.containerRecommendations.lowerBound**
      object

      Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

    - **status.recommendation.containerRecommendations.scope**
      string

      Scope binds this recommendation to a specific scope value in form “ =".

    - **status.recommendation.containerRecommendations.target**
      object

      Required value

      Recommended amount of resources. Observes `ContainerResourcePolicy`.

    - **status.recommendation.containerRecommendations.uncappedTarget**
      object

The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`, including `recommendationFactor`. May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

- **status.recommendation.containerRecommendations.upperBound**
  object

  Maximum recommended amount of resources. Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

VerticalPodAutoscalerCheckpoint

Short names: vpacheckpoint

Scope: Namespaced

- v1
- v1beta2

VerticalPodAutoscalerCheckpoint is the checkpoint of the internal state of VPA that is used for recovery after recommender's restart.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Specification of the checkpoint. More information in the documentation.

  - **spec.containerName**
    string

    Name of the checkpointed container.

  - **spec.vpaObjectName**
    string

    Name of the VPA object that stored VerticalPodAutoscalerCheckpoint. object.

- **status**
  object

  Data of the checkpoint.

  - **status.cpuHistogram**
    object

    Checkpoint of histogram for consumption of CPU.

    - **status.cpuHistogram.bucketWeights**
      object

      Map from bucket index to bucket weight.

    - **status.cpuHistogram.referenceTimestamp**
      string

      Reference timestamp for samples collected within this histogram.

Reference timestamp for samples collected within this histogram.

- **status.cpuHistogram.totalWeight**
  number

  Sum of samples to be used as denominator for weights from BucketWeights.

- **status.firstSampleStart**
  string

  Timestamp of the fist sample from the histograms.

- **status.lastSampleStart**
  string

  Timestamp of the last sample from the histograms.

- **status.lastUpdateTime**
  string

  The time when the status was last refreshed.

- **status.memoryHistogram**
  object

  Checkpoint of histogram for consumption of memory.

  - **status.memoryHistogram.bucketWeights**
    object

    Map from bucket index to bucket weight.

  - **status.memoryHistogram.referenceTimestamp**
    string

    Reference timestamp for samples collected within this histogram.

  - **status.memoryHistogram.totalWeight**
    number

    Sum of samples to be used as denominator for weights from BucketWeights.

- **status.totalSamplesCount**
  integer

  Total number of samples in the histograms.

- **status.version**
  string

  Version of the format of the stored data.

Deprecated resource. Support for the resource might be removed in a later release.

VerticalPodAutoscalerCheckpoint is the checkpoint of the internal state of VPA that is used for recovery after recommender’s restart.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Specification of the checkpoint. More information in the documentation.

  - **spec.containerName**

string

Name of the checkpointed container.

- **spec.vpaObjectName**
  string

  Name of the VPA object that stored VerticalPodAutoscalerCheckpoint. object.

- **status**
  object

  Data of the checkpoint.

  - **status.cpuHistogram**
    object

    Checkpoint of histogram for consumption of CPU.

    - **status.cpuHistogram.bucketWeights**
      object

      Map from bucket index to bucket weight.

    - **status.cpuHistogram.referenceTimestamp**
      string

      Reference timestamp for samples collected within this histogram.

    - **status.cpuHistogram.totalWeight**
      number

      Sum of samples to be used as denominator for weights from BucketWeights.

  - **status.firstSampleStart**
    string

    Timestamp of the fist sample from the histograms.

  - **status.lastSampleStart**
    string

    Timestamp of the last sample from the histograms.

  - **status.lastUpdateTime**
    string

    The time when the status was last refreshed.

  - **status.memoryHistogram**
    object

    Checkpoint of histogram for consumption of memory.

    - **status.memoryHistogram.bucketWeights**
      object

      Map from bucket index to bucket weight.

    - **status.memoryHistogram.referenceTimestamp**
      string

      Reference timestamp for samples collected within this histogram.

    - **status.memoryHistogram.totalWeight**
      number

      Sum of samples to be used as denominator for weights from BucketWeights.

  - **status.totalSamplesCount**
    integer

    Total number of samples in the histograms.

  - **status.version**
    string

    Version of the format of the stored data.

> **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**
>
> The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

FAQ

> The module lifecycle stage: General Availability
>
> The module has requirements for installation

How do I view the Vertical Pod Autoscaler recommendations?

You can view the VPA recommendations after the VerticalPodAutoscaler custom resource is created using the following command:

```
d8 k describe vpa my-app-vpa
```

The `status` will have the following parameters:

- `Target` — the optimal amount of resources for the Pod (within the resourcePolicy).
- `Lower Bound` — the minimum recommended amount of resources for the regular operation of the application.
- `Upper Bound` — the maximum recommended amount of resources. Most likely, the resources above this upper bound will never be used by the application.
- `Uncapped Target` — the target computed from resource usage only, without the `resourcePolicy` constraints (`minAllowed`, `maxAllowed`) and without `recommendationFactor`.

How does Vertical Pod Autoscaler handle limits?

Example No. 1

The following example shows a VPA object:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"
```

The VPA object contains a Pod with the following resources:

```
resources:
  limits:
    cpu: 2
  requests:
    cpu: 1
```

If a container uses all the CPU, and VPA recommends 1.168 CPU for that container, then the ratio between requests and limits will be 100%. In this case, when recreating the Pod, VPA will modify it and set the following resources:

```
resources:
  limits:
    cpu: 2336m
  requests:
    cpu: 1168m
```

Example No. 2

The following example shows a VPA object:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"
```

The VPA object contains a pod with resources:

```
resources:
  limits:
    cpu: 1
  requests:
    cpu: 750m
```

If the request-to-limit ratio is 25% and VPA recommends 1.168 CPU for the container, VPA will change the container resources as follows:

```
resources:
  limits:
    cpu: 1557m
  requests:
    cpu: 1168m
```

If you need to limit the maximum number of resources that can be allocated to container constraints, you should use `maxAllowed` in the VPA object specification or use the Limit Range of the Kubernetes object.

How do I make VPA recommend more or less than it measured?

VPA computes the recommendation from the measured resource usage and adds its own safety margin. To make VPA recommend more or less, set the `recommendationFactor` field in the container policy of the VerticalPodAutoscaler object. VPA multiplies the computed recommendation by the specified value. For example, `1.2` increases the recommendation by 20%, and `0.87` decreases it by 13%.

Set the multipliers in the VerticalPodAutoscaler object. A policy with `containerName: "*"` applies to all containers that have no policy of their own:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  resourcePolicy:
    containerPolicies:
    - containerName: "*"
      recommendationFactor:
        cpu: "1.2"
        memory: "0.87"
```

The `cpu` and `memory` values are quantities written as strings with at most two decimal places, for example `"1.25"`. The allowed range is greater than zero and up to `10` inclusive. If you omit a key, the recommendation for that resource does not change.

VPA applies the multiplier after its own safety margin and before `minAllowed` and `maxAllowed`, so these constraints still limit the final recommendation. In the `status` of the VPA object, `target`, `lowerBound` and `upperBound` already include the multiplier. `uncappedTarget` shows the target computed without it.

Limits follow requests proportionally, as described in the section How does Vertical Pod Autoscaler handle limits?.

⚠ A `memory` multiplier below `1` increases the risk of OOM kills, because it reduces the headroom VPA keeps above the measured usage.

Changing a multiplier changes the recommendation, so the `Updater` component can update the Pods of the application according to its usual rules.

For a DaemonSet with `spec.scope`, VPA stores the recommendations in `status.groups` in a compact form that contains only `target` for each container. The value without the multiplier is not shown there.

i **Starting from version 1.76, Deckhouse Kubernetes Platform has been renamed to Deckhouse Platform.**

The platform now offers more features, and editions have changed. We are already updating the documentation, but if you have any questions, please contact your account manager or technical support.

Examples

i The module lifecycle stage: General Availability

The module has requirements for installation

The module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
```

The basic `VerticalPodAutoscaler` custom resource example

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-app
```

The advanced `VerticalPodAutoscaler` custom resource example

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "InPlaceOrRecreate"
  resourcePolicy:
    containerPolicies:
    - containerName: hamster
      minAllowed:
        memory: 100Mi
        cpu: 120m
      maxAllowed:
        memory: 300Mi
        cpu: 350m
      mode: Auto
```